GBS Software Architecture

Version 2.1

February 3, 1997

CDRL SEQUENCE NO. A004

Sponsored by:

Defense Advanced Research Projects Agency
Information Systems Office
DARPA Order No. E456
Issued by DARPA/CMO under Contract MDA972-96-C-0025

Prepared by:

Welkin Associates, Ltd.
Suite 410
10300 Eaton Place
Fairfax, Virginia 22030
(703) 691-4616

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	3 February 1997	Final Report

4. TITLE AND SUBTITLE
GBS Software Architecture Design Document, Version 2.1

5. FUNDING NUMBERS
MDA972-96-C-0025

6. AUTHOR(S)
M. Snellings, M. Giboney, K. Dezell, J. Nyikos

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Welkin Associates Ltd.
10300 Eaton Place Suite 410
Fairfax, Va. 22030

8. PERFORMING ORGANIZATION REPORT NUMBER
0025-CDRL-004

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
Information Systems Office
3701 North Fairfax Drive
Arlington, Va 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
None

12a. DISTRIBUTION AVAILABILITY STATEMENT
Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document describes the software architecture for Version 2.1 of the Global Broadcast Service (GBS). This architecture only addresses the transmission and receipt of file, stream, and status data which comprises a subset of the channels broadcast over the GBS system. Throughout the remainder of this document, the terms 'GBS' and 'GBS system' are used to mean the portion of the GBS broadcast which transmits file, stream, and status data.

14. SUBJECT TERMS
GBS Software Architecture Version 2.1

15. NUMBER OF PAGES
51

16. PRICE CODE
N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Table of Contents

1. Overview 3

1.1 Global Broadcast Service 3
1.2 Joint Broadcast Service 3
1.3 Document Overview 3
1.4 Additional Supporting Documents 3

2. Software Architecture 4

2.1 General Configuration 4
2.1.1 GBS Directory Structure 5
2.1.2 File Wrapper 9
2.1.3 GBSID 10
2.1.4 Destination File 10
2.1.5 Destination Groups 11
2.1.6 GBS File Types 11
2.1.7 Keyword File 11
2.1.8 Process Management 11
2.2 GBS Data Source 14
2.2.1 Simplified Data Source 16
2.3 Broadcast Management Center (BMC) 16
2.3.1 Gateway 16
2.3.2 Queue Manager 17
2.3.3 Broadcast Status 18
2.3.4 Broadcast Stream Data 19
2.3.5 Crypto Synchronization 21
2.4 GBS Receive 21
2.4.1 File Receipt and Storage 22
2.4.2 Receive Broadcast Status 24
2.4.3 Receive Stream Data 25
2.4.4 File Management 26
2.4.5 Executive 30

Appendix A GBS Programs 31

Appendix B GBS Shell Scripts 35

Appendix C Generic Configuration Parameters 37

Appendix D Executive Configuration File 40

1. Overview

1.1 Global Broadcast Service

The Global Broadcast Service (GBS) seeks to take advantage of commercial communications technology to provide intelligence data to the intelligence consumer in a timely fashion. It is important to note that GBS itself is NOT an intelligence system, but the mechanism to provide intelligence information to other users, either human or intelligence exploitation systems. Of particular importance is the fact that GBS can provide LARGE amounts of audio, video, and data, up to 23 Mbps, to locations which previously could not receive such information due to physical or bandwidth constraints. This large bandwidth can be achieved at a fraction of the cost of custom built communications system with a minimal set of hardware. High bandwidth with minimal hardware constraints opens the door for providing intelligence information to the warfighter in the lower echelons. It also means that intelligence such as video and imagery which may have been previously unavailable to the warfighter may now be provided.

1.2 Joint Broadcast Service

The Joint Broadcast Service (JBS) is a fielded operational version of the GBS system which supports Operation Joint Endeavor. JBS is a video and data delivery service for theater requested information provided by numerous in-theater and CONUS sources.

The Joint Information Management Center (JIMC), located at the Pentagon, coordinates the data broadcast by JBS. JBS uses a leased Orion 1 satellite transponder to broadcast a 23 Mbps data stream containing several audio/video channels (CNN Headline News, Armed Forces Radio Television System (AFRTS), and UAV video) and several data channels (IP US Secret data, IP NATO data, ATM US Secret data, ATM NATO data). The data products are transmitted from Washington, D.C. to the high-powered tranponder on the Orion satellite for relay to NATO and IFOR forces deployed over a large geographic area (including Bosnia, Italy, Germany, Hungary, Belgium, England, and several ship based systems).

1.3 Document Overview

This document describes the software architecture for Version 2.1 of the Global Broadcast Service (GBS). This architecture only addresses the transmission and receipt of file, stream, and status data which comprises a subset of the channels broadcast over the GBS system. Throughout the remainder of this document, the terms 'GBS' and 'GBS system' are used to mean the portion of the GBS broadcast which transmits file, stream, and status data.

Readers of this document should acquire a high level understanding of the software as well as specific details required for troubleshooting and maintaining the GBS system. The history of the GBS software releases to date are described below:

Version 1.01 - First deployed JBS release (March 1996)
Version 1.06 - JBS software upgrade to fix bugs and add minor features (June 96)
Version 2.03 - JWID 96 release (Aug 96) (deployed in CONUS only)
Version 1.07 - JBS ATM deployment release (Sept 96)
Version 2.1 - JBS software upgrade (Jan 97)

1.4 Additional Supporting Documents

The following documents may provide additional information:

GBS Version 2.1 Receive System Training and User's Manual
GBS Version 2.1 Release Notes
GBS Software Installation Instructions
MTN Queue Manager Reference Manual
GBS File Wrapper Version 3.0

2. Software Architecture

There are three basic architectural components of the GBS system. The first is the GBS data source which is a provider of the data files to be included in the broadcast. The second is the broadcast management center (BMC) which takes data files from the source and includes them in the broadcast uplink. The third component of GBS is the receive site where data from the broadcast is accepted and processed. Figure 1 High Level GBS Architecture shows the basic GBS architecture for data file processing.

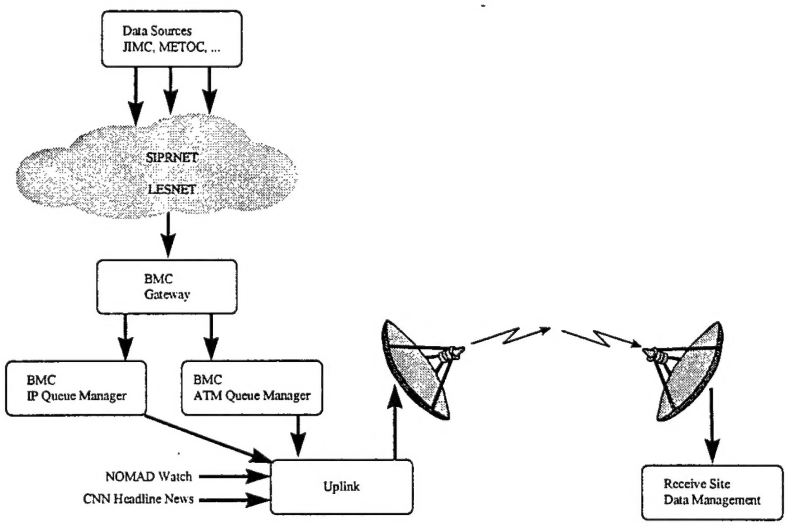


Figure 1 High Level GBS Architecture

The GBS hardware configuration supports both IP and ATM protocols. These protocols define the physical connectivity between the BMC workstation and satellite transmitter, and the satellite receiver and Receive workstation. The data broadcast over each channel (e.g., ATM US Secret data) uses a single protocol; the channel includes file, stream, and status data combined in single data channel. The software used to support the IP and ATM transport protocols is identical, with several run time configuration file differences. Except for the transport programs that directly inject data into and retrieve data from the broadcast (the same programs use different protocols based on a run time parameter), the GBS system is independent of the transport protocol. Therefore, this document is applicable to both an IP and ATM, with the few protocol specific differences noted.

2.1 General Configuration

This section describes portions of the GBS software architecture that are common across the three major architectural areas.

2.1.1 GBS Directory Structure

GBS relies on a pre-defined directory structure. This directory structure is created when the software is installed and the root of this structure is referred to as $GBS_HOME for the remainder of this document. The directory structure contains the GBS software executables, configuration files, data files, logs which show the status of the system, and files which are used internally by the system. The GBS software accesses programs, configuration files, and data files through both the environment variable $GBS_HOME and the directory /home/GBS (which is normally managed by the automounter daemon) - these two should always refer to the same physical point in the filesystem. Table 1 GBS Directory Structure shows the directory structure and how the directories are used in the three major architectural areas of the system. This table uses the symbol ✓ to indicate that the directory is used by the specified architectural component; the symbol ✗ is used when the architectural component does not use the directory.

Table 1 GBS Directory Structure

Directory under $GBS_HOME and Purpose	Source	BMC	Receive
app-defaults Contains X Windows application default files (the XAPPLRESDIR environment variable points to this directory). All GBS programs (located in the $GBS_HOME/bin directory) use the application default file named GBS to set their X Windows resources. This directory could contain other application default files (e.g., for the Matrix application).	✗	✗	✓
applinks Ties applications used to analyze files received over the broadcast to file extensions and file types through the use of symbolic links. This is analogous to setting mime types in the mailcap file.	✗	✗	✓
apps Scripts that initialize an environment and then starts a data exploitation application (usually located in /home/gbsapps or /opt).	✗	✗	✓
bin GBS developed programs	✓	✓	✓
comms Files for named pipes (UNIX sockets) used for inter-process communication	✓	✓	✓
config Run time configuration files	✓	✓	✓
customFiles Used only during software installation; files from this directory are copied to other locations. Customization files are included for Netscape and http.	✓	✓	✓
data Files received over the broadcast are saved under this directory, normally in a subdirectory which is named to indicate the type of data.	✗	✗	✓
email Previously used to support file submission through email; this feature may be re-enabled in a future release.	✗	✗	✗
header Top level directory not explicitly used	✗	✗	✓
header/current Wrappers of each file received over the broadcast. Wrappers are deleted when the corresponding Data Manager record is removed.	✗	✗	✓

Directory under $GBS_HOME and Purpose	Source	BMC	Receive
header/dat Contains data files used by the Data Manager. The Data Manager accesses these files through mmap(), so they must be physically located on the machine where *dmServer* runs.	✗	✗	✓
header/discard No longer used	✗	✗	✗
header/error Wrappers of files that were received over the broadcast but could not be registered with the Data Manager (for some reason other than the connection between *dispose* and *dmServer* was down).	✗	✗	✓
header/new Top level directory not explicitly used. File wrappers are temporarily stored under this directory if *dispose* cannot communicate with the Data Manager. As soon as this communication path is re-established, *dispose* will register these wrappers with the Data Manager and then move the wrappers into the header/current directory.	✗	✗	✓
header/new/deleted Wrappers of files received over the broadcast and saved but immediately deleted after processing, so treated as though the file was deleted. This is only used for `system.thl` files.	✗	✗	✓
header/new/discarded Wrappers of files received over the broadcast but not saved.	✗	✗	✓
header/new/error Wrappers of files that were received over the broadcast which could not be processed any further.	✗	✗	✓
header/new/saved Files received over the broadcast and saved.	✗	✗	✓
header/new/tmp Temporary working directory for the *dispose* process. This is used to completely write out a saved file, set the file's date and access permissions before the file is moved into the save directory.	✗	✗	✓
help Contains help files for GUI applications.	✓	✓	✓
lib Runtime libraries required by the GBS software.	✓	✓	✓
logs Log files showing status of the system. This is normally a symbolic link pointing to /var/GBS_logs	✓	✓	✓
logs/THL Summary of files sent out over the broadcast in last X minutes where X is configurable, but set to 15 minutes by default.	✗	✓	✗
must Location where files wrapped with the must receive flag are saved if they do not match any of the dispose rules.	✗	✗	✓
oil_user Contains default user customization files for Oilstock. Used when new accounts are setup to use GBS.	✗	✗	✓

Directory under $GBS_HOME and Purpose	Source	BMC	Receive
queues/ATM/0-4 **queues/IP/0-4** Queued priority directories (5 per queue) containing symbolic links to wrapped files currently queued for broadcast. The directories correspond to file priority where 0 is the highest priority (FLASH IMMEDIATE) and 4 is the lowest (ROUTINE).	✗	✓	✗
queues/ATM/new **queues/IP/new** A symbolic link to the wrapped file is placed in this directory when a file is placed into the broadcast queue. The queue manage will move the symbolic link from the new directory into the proper queued priority directory.	✗	✓	✗
rdm_html No longer used	✗	✗	✗
receive Top level directory not explicitly used.	✗	✗	✓
receive/.bad_blocks No longer used	✗	✗	✗
receive/.bad_received Contains files with named with the GBS ID of files that were not completely received over the broadcast (only error tolerant files are represented). Each file is 4 bytes, containing an integer reflecting the number of missing bytes in the data file. This is used by the transport programs to attempt to receive the file again if it is re-broadcast.	✗	✗	✓
receive/.dsa_assem Temporary working directory where files being received over the broadcast are assembled.	✗	✗	✓
receive/.received Zero length files named by the GBS ID for files that were successfully received over the broadcast. This is used to indicate that subsequent transmissions of this file can be ignored. This directory is automatically cleaned; files remain for approximately 2 days.	✗	✗	✓
receive/bad_receive Incomplete wrapped files that have been received over the broadcast are moved to this directory for *dispose* after all expected transmissions are complete. Any files that do not have their error tolerant flag turned on in the wrapped are immediately discarded.	✗	✗	✓
receive/bad_wrapper No longer used	✗	✗	✗
receive/corrupt_files Wrapped files are placed in this directory if *dispose* is unable to process the file because it cannot read the wrapper.	✗	✗	✓
receive/error_receive No longer used	✗	✗	✗
receive/good_receive Wrapped files are moved to this directory for *dispose* after they have been completely received.	✗	✗	✓
receive/retranReq Zero length files named by the GBS ID for indicating that the file has been requested for retransmission and therefore should be saved when it is received (regardless of the dispose rules).	✗	✗	✓

Directory under $GBS_HOME and Purpose	Source	BMC	Receive
rtx Top level directory not explicitly used.	✗	✗	✓
rtx/autoIn No longer used	✗	✗	✗
rtx/manualIn No longer used	✗	✗	✗
rtx/reqGen No longer used	✗	✗	✗
rtx/reqGen/reqSend No longer used	✗	✗	✗
rtx/rexmitDiag No longer used	✗	✗	✗
scripts GBS scripts much like the bin directory.	✓	✓	✓
skel GBS specific environment files used to create new GBS users and to give existing users access to the GBS System.	✓	✓	✓
status Contains last status messages sent from the BMC.	✗	✓	✗
system Received files wrapped with type "system.*" are saved in this directory. If the file was wrapped with a subtype of system, then the subtype is converted to a directory where the file will be saved.	✗	✗	✓
tlfDir Keeps overall summary of files sent out over the broadcast.	✗	✓	✗
unwrapped Top level directory not explicitly used.	✗	✓	✗
unwrapped/notify Contains a file that contains a path to the file to be automatically wrapped. A default wrapper file must also be provided in the same directory as the file to be wrapped. The default wrapper file must be called defaults.gbs.	✗	✓	✗
wrapped Contains wrapped files that will be sent on the broadcast. On a source system, the files will be sent to the BMC and deleted from this directory. At the BMC, these files are included in the broadcast. The directory needs to be manually cleaned periodically.	✓	✓	✗
wrapped/archive Contains files that were wrapped with an archival request.	✗	✓	✗
wrapped/errors Zero length files named by the GBS ID indicating an error was encountered in processing the file.	✗	✓	✗
wrapped/notify Zero length files named by the GBS ID indicating that the corresponding file in the wrapped directory is complete.	✓	✓	✗
wrapped/reachback Not used, planned future capability.	✗	✓	✗
wrapped/save No longer used	✗	✗	✗

Directory under $GBS_HOME and Purpose	Source	BMC	Receive
wrapped/timed/.timed No longer used	✗	✗	✗
wrapped/timed/replacable Indicates that a file being scheduled for inclusion into the broadcast can be replaced by another file.	✗	✓	✗
wrapped/timed/timed Indicates that a file in wrapped is to be scheduled for inclusion into the broadcast.	✗	✓	✗
wrapped/transmit Indicates files in the wrapped directory should be passed to the queue manager.	✗	✓	✗
www Top level directory not explicitly used.	✓	✓	✓
www/authdocs HTML documents and supporting files such as the password file that allow wrapping of files through a Web browser.	✗	✓	✗
www/cache No Longer Used (?)	✗	✗	✗
www/cgi-bin CGI programs. Source: used to create Intelink X pages. Receive: used to support Web based file management and Intelink X.	✓	✗	✓
www/cgi-bin.upload CGI programs for wrapping files through a Web browser.	✗	✓	✗
www/cgi-bin.upload/template Template HTML files for wrapping files through a Web browser.	✗	✓	✗
www/Default_Config No longer used	✗	✗	✗
www/htdocs HTML home page.	✗	✗	✓
www/images Images used in HTML pages.	✓	✓	✓
www/orderforms Location where Intelink X source order pages are saved when received.	✗	✗	✓
www/Upload Contained directories with IP addresses of the receive station (?)	✗	✓	✗

2.1.2 File Wrapper

Most message passing protocols add control information to the data that is being passed from the originator to the destination. The type of control information and its use vary depending on the protocol. GBS also follows this paradigm for the data files that are included in the broadcast. Each data file has a header called the wrapper added to it prior to the transmission. When received, the wrapper is removed and the file appears as it did on the send side.

The information within the wrapper is used for several functions. The wrapper information provides important processing information used by the software during the steps of the broadcast. This information is typical of most header information found in other message processing protocols in that it may not be of use to the end user but is instrumental in allowing the software to function correctly. Examples of how the wrapper fields can be used to affect processing are listed in Table 2 Example Wrapper Fields.

Table 2 Example Wrapper Fields

Wrapper Field	Use
destination	Indicates which destinations should receive the file, indirectly specifying which transport mechanism (IP or ATM) to use when sending the file.
file type	Indicates if a particular broadcast address (broadcast or multicast) or VP/VC is to be used when sending the file. Primary means of filtering and organizing files at the receive sites.
priority	Files with a higher priority are sent over the broadcast before files of lower priority.
size	Specifies the number of bytes to be sent and allows the broadcast to know when a transfer is complete.

At the receive sites, the primary function of the wrapper is to provide the receivers of the files with a means of determining whether or not that particular receive site wishes to keep or discard the files being received over the broadcast. A process called *dispose* compares the contents of each file wrapper against a set of rules to determine whether the file should be kept and if so where on the disk it should be saved. The *dispose* process is discussed in section 2.4.1.

A third use of information from the wrapper is to allow the user to add information to the file, and to give summary information about the file transfer to the receive sites. By adding fields to the wrapper, the user is able to convey additional information about the file to the receivers and can specify actions to be taken during the steps required to broadcast the file.

Another usage of information from the wrapper is to give the user and maintainers of the system valuable tracking information such as which source sent the file, the date the file was sent, and the GBS gateway that processed the file.

For more information on the GBS file wrapper see the GBS File Wrapper document.

2.1.3 GBSID

Systems that cause files to be included into the GBS broadcast are called sources. The system that receives data from the sources and processes them for inclusion into the GBS broadcast is called the GBS gateway. The GBS gateway can process files from many sources and can itself act as a source. GBS does not enforce any restrictions on the file management for each source. Therefore, it is possible for two different sources to send a file with the same name to the GBS gateway for inclusion into the GBS broadcast.

In order to handle files with duplicate names and for internal tracking purposes, each file that is transmitted over the GBS broadcast is assigned a unique GBS identifier (GBSID). This identifier is made up of two parts. The first part contains the host name of the computer that is the source of the file (up to the first 8 characters). The second part contains eight characters that represent the hexadecimal representation of a one up number that is maintained for each source. For example, if a computer at the AIA source has a host name of "aiaSunWorkstation", the first file transmitted over the GBS broadcast from AIA would have a GBSID of aiaSunWo00000000. The second file transmitted from AIA would have a GBSID of aiaSunWo00000001 etc. Due to the cryptic nature of the GBSID, it is hidden from the end user but can be useful to maintenance personnel in troubleshooting problems.

2.1.4 Destination File

The file $GBS_HOME/config/destList contains a list of known locations that are receiving the broadcast. GBS is a true broadcast system which means that ALL receivers listening to a queue can receive all of the data that is sent out over that broadcast queue, although some of the data can be filtered at the network level by not listening to IP multicast channels or ATM VP/VCs. However, destinations that are in the destination file can be added to a wrapper to give the receive locations a means of looking for specific data

or screening out data that is of no interest. Again, adding destinations to the wrapper does NOT prevent the file from being delivered to the destinations that are not specified.

Each destination in the destination file also indicates which queues a file should be broadcast over to reach the particular destination. GBS version 2.1 automatically adds the destination "all" to each of the files wrapped for inclusion in the broadcast. The default destination file specifies that the destination "all" is to be sent over both the IP and ATM queues. Therefore, by default, GBS version 2.1 sends all files over both the IP and ATM transport mechanisms. Besides individual sites, groups can also be listed. Each group listed must also specify one or more queues and the keyword 'GROUP'.

2.1.5 Destination Groups

The directory $GBS_HOME/config/group contains files which group destinations from the destList file into groups. The purpose for groups is to logically group sites based on a need for a common type of data. Possible groupings might be based on geography, service branch, or function (e.g., J2, J3, etc.). Groups then provide a convenient way of assigning multiple destinations to a file within the GBS file wrapper. The file name of each file in the directory represents the name of a group. Each of these files contains the names of the destinations that are part of the group.

The receive sites have control over the groups in which they belong. Each site registers locally to belong to one or more groups. A site is encouraged, but not required, to notify the BMC when they change group membership. Therefore, a source site should not rely on their assumption of who belongs to a group.

2.1.6 GBS File Types

The file $GBS_HOME/config/typeList contains a list of current GBS file types. File types provide a means of categorizing the files that are sent across the GBS broadcast. These file categories can then be used by the receivers of the broadcast to determine which file types should be saved, where they should be saved, and which types should be discarded. File types also provides a convenient means of viewing the data that has been received over the broadcast. GBS file types can be segmented into sub-types to form type hierarchies. For example, an imagery file type could be broken into primary and secondary sub-types. Each of these subtypes could then be broken down further into the type of imagery such as NITF, JPEG etc. Each file transmitted across the GBS broadcast is required to have one of these types specified in the wrapper.

2.1.7 Keyword File

The file $GBS_HOME/config/keywords contains a list of current GBS keywords. Keywords again provide a way for the receive locations to look for specific files of interest or to screen out files that are not of interest.

2.1.8 Process Management

All background (daemon) GBS UNIX programs are managed by a controlling program called the *processMonitor*. The *processMonitor* provides feedback on the status of the system, keeps daemon programs running, restarts programs if they terminate, notifies the user when critical programs are down, and schedules programs to run periodically or at specific times. A single *processMonitor* program runs on a GBS machine. The *processMonitor* determines which programs to control and how to control those programs by reading configuration files. The configuration files read by the process monitor are passed as command line arguments following the option -f as shown below. Multiple configuration files may be passed to the *processMonitor* by specifying multiple -f options.

```
processMonitor -f processFile [-f processFile]
```

Each configuration file uses key value pairs to describe the programs that are to be monitored by the *processMonitor*. Table 3 Process File Options shows the valid key value pairs that can be used within a configuration file. Lines in a configuration file beginning with # are comments.

Table 3 Process File Options

Parameter	Description
PROGRAM:	Defines the executable file to be run. This can be a Bourne shell script, Perl shell script, C program, etc. The only known thing that it can not be is a C shell script.
NAME:	Name of the process which should be understandable by the end user of the system.
ARGUMENTS:	Specifies arguments to be passed to the executable. All Standard arguments such as -f filename are supported. Output redirection using >, >!, >> etc. are not supported. Output is redirected via the keywords SDT_OUT: and STD_ERR:.
STATIC:	No value required for this key. This indicates that the program should always be kept running.
START_EVERY_N_MINUTES:	Indicates that the program is to be run periodically (Positive Integer).
SCHEDULE:	Specifies a schedule of times when the executable is to be run. This is a string similar to a cron entry. The string must contain five SPACE separated integer patterns: minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), day of the week (0-6 with 0=Sunday). Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). Both are adhered to if specified as a list of elements. See the man page on crontab for examples.
CONDITIONAL:	Indicates that the program may or may not be started based on the return value from the specified executable. If the specified executable has a positive exit status (1) the program will be started. Otherwise the program will not be started.
MAX_RESTARTS:	An integer specifying the maximum number of times to restart the program. This overrides the default value of 10 for a static programs.
CRITICAL:	No value required for this key. Indicates that it is critical that the static program remain running. If this program terminates a system alert will be issued. If it terminates and exceeds the maximum allowed restarts, a system alert will be generated every 5 seconds.
STD_OUT:	Specifies a file to which output that would normally go to the standard out will be written.
STD_ERR:	Specifies a file to which output that would normally go to standard error will be written.
INIT_EXEC:	Specifies an executable to be run before the program will be run. This allows initialization required by the program to be performed, though it is strongly recommended that the program perform its own initialization.

The definition of a program to monitor begins with a line beginning with PROGRAM:. The definition continues until another PROGRAM: line or the end of the file is encountered. Listed below are rules regarding the program specifications:

- The NAME: and PROGRAM: lines are required for each process.
- The options STATIC: and START_EVERY_N_MINTUES: are incompatible.
- The options STATIC: and SCHEDULE: are incompatible.
- The options SCHEDULE: and START_EVERY_N_MINUTES: are incompatible.
- MAX_RESTARTS: is only applicable when the STATIC: option is also specified.
- CRITICAL: is only applicable when the STATIC: option is also specified.

If any errors are encountered in reading a configuration file the program where the error was encountered will not be managed by the *processMonitor*. Errors can occur when invalid data is entered in the file such as a line that reads

START_EVERY_N_MINTUES: YES

where an integer was not provided or if any of the above mentioned incompatible combinations of options are specified.

The program *pmDriver* provides insight into the *processMonitor* and to allow the user to send commands to the *processMonitor* affecting how programs are managed . This program is provided for convenience only; it is not necessary for the normal operation of the GBS software. The *pmDriver* program is a menu driven command line program. When *pmDriver* is started, it requests that the user enter the host name of the computer where *processMonitor* is running and then presents the user with a menu. Table 4 pmDriver Menu describes the menu options.

Table 4 pmDriver Menu

Menu Item	**Description**
System Status Request	Sends a system status request to the *processMonitor* and displays the results.
Restart Stopped Static Process	Commands the *processMonitor* to restart a static program that is no longer running or being restarted since it has reached the maximum number of restarts.
Cycle Static Process	Commands the *processMonitor* to terminate and then restart the specified static program.
Stop Managing Process	Commands the *processMonitor* to terminate and no longer manage the specified program.
Read Process File	Commands the *processMonitor* to read the specified configuration file and begin managing any programs that are in the file (any programs already being managed are ignored).
Replace Process From File	Commands the *processMonitor* to terminate the specified program and begin managing the specified program using the options from the specified process file.
Run Periodic Process Now	Commands the *processMonitor* to run a scheduled or periodic program immediately rather than at the next scheduled execution time.

2.2 GBS Data Source

Each GBS data source that provides data files to GBS for inclusion in the broadcast must install the GBS source software and have network connectivity to the BMC, in particular the to GBS gateway machine. A data source normally supplies files to a single gateway. However, the software does support sending files to multiple gateways. The current GBS gateways are listed in Table 5 GBS Gateways. Furthermore, each source must contact the GBS gateway support personnel to obtain a login to the GBS gateway machine. This login is a limited account which will allow the source to transfer the files for inclusion in the broadcast to the GBS gateway machine and will not allow the users to login to the GBS gateway machine. If the source will be wrapping files through the Web interface, the source must also be granted access to the HTML and CGI programs on the GBS gateway that provide the Web wrapping functionality.

Table 5 GBS Gateways

IP Address of GBS Gateway	Description
199.55.96.180	BC2A US Secret Gateway (located at the Pentagon)
199.55.96.183	BC2A NATO Gateway (located at the Pentagon)
199.55.96.181	GBS Testbed US Secret Gateway (located at the Pentagon)
	JITI Mobile Gateway

Once the source computer has been configured, the user may send data files over the broadcast by selecting the files to send and adding the GBS wrapper. The GBS wrapper can be added interactively through the use of a Graphical User Interface (GUI) program called *wrap* or by running an interactive command line program called *wrapit*. The *wrapit* program allows multiple files to be wrapped by specifying the wrap option on the command line or within a default wrap file. Use of the *wrapit* program can be automated in a script or by other means if the source wishes to automatically wrap files. The *wrapit* program has the following syntax:

wrapit [-d <defaultFile>] [options] <filename>

The options are listed in Table 6 wrapit Options. Reference the GBS Wrapper document for additional information regarding the setting of the values.

Table 6 wrapit Options

Option	Allowed Value
-P priority	integer 1=FLASH, 2= OP IMEMDIATE, 3=PRIORITY, 4=ROUTINE
-D dest	ascii string - can be more than one -D entry
-t type	ascii string, the type must match an entry in the typeList configuration file
-r	must_receive
-c file_markings	ascii string
-p permissions	octal int
-A	ascii
-C	make copy
-i file_info	ascii string (single line)
-w signature	ascii string (single line)
-q qulaity_of_service	h=high, m=medium, l=low
-F frequency_of_retrans	int (minutes)
-L late_after	ddhhmm[z] mon yy
-b start_at	ddhhmm[z] mon yy
-l relative_location	ascii scring

Option	Allowed Value
-m mail_on	bitwise or of the following: 1 = when file is done being transferred 2 = on error free receipt 4 = on receipt with errors
-e e_mail	ascii string
-k archive_days	int
-E	error_tolerant
-I replaces	ascii string
-B BENumber	ascii string
-o expires	ddhhmm[z] mon yy
-K keyword	ascii string
-U name:val	ascii string

Once the files have been wrapped, a daemon program called *xferit* uses ftp to send the files to the BMC. Figure 2 GBS Data Source Architecture shows the architecture of a GBS data source supplying files to be included in the GBS broadcast.

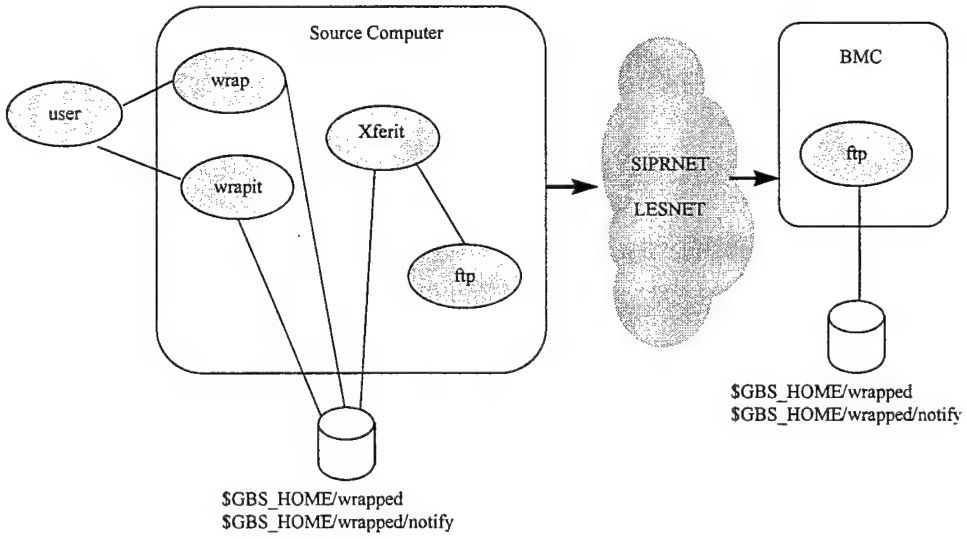


Figure 2 GBS Data Source Architecture

The *xferit* program periodically polls the $GBS_HOME/wrapped/notify directory to determine whether any files need to be transferred to the gateway. If any new files are found, then *xferit* will invoke ftp (which requires that the $GBS_HOME/.netrc file be configured to log into the gateway without a password and change into the correct directory). Once the file is transferred, *xferit* will delete both the wrapped and notify files.

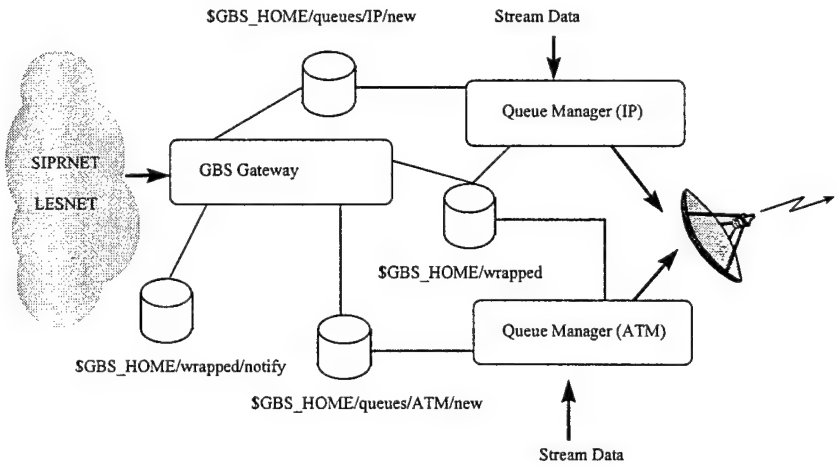
2.2.1 Simplified Data Source

A site can also forward files to be included in the broadcast without installing the GBS software locally. A site can obtain an ftp login at the gateway and work with an operator at the gateway to set up a defaults file. Then the site ftps a file to the gateway and it is wrapped based on the defaults file and included in the broadcast.

2.3 Broadcast Management Center (BMC)

The broadcast management center has four main functions. The first is to accept files from the data sources for inclusion into the GBS broadcast. This function is performed by the Gateway. The second function is to queue the files received from the data sources to the appropriate transport. This is performed by the Queue Manager in close concert with the Gateway. There is a queue manager for each of the supported transport mechanisms currently ATM and IP. The Queue Managers and the Gateway must share the same $GBS_HOME/wrapped and $GBS_HOME/queues directories, although the Gateway and Queue Manager functionality need not exist physically on the same computer. The third function is to provide an indication to the receive terminals about the status of the broadcast. The fourth function is to add stream data to the broadcast. Figure 3 BMC Architecture shows the architecture of the BMC.

Figure 3 BMC Architecture



2.3.1 Gateway

The Gateway receives files from the GBS data sources, determines the type of transport required (ATM or IP), and passes the files to the appropriate queue managers for inclusion into the GBS broadcast. Files received from the sources can either be wrapped files ready for transmit or files that have not yet been wrapped. Files that have not yet been wrapped will be wrapped by the *autowrap* program. The GBS file wrapper is parsed to determine any special processing required for the file such as periodic transmission. Various fields within the wrapper are updated to show the processing steps that have been completed. The type of transport required is determined from the wrapper based on the destinations. Each destination in the wrapper is mapped to one or more transport mechanisms via the file described in Destination File.

GBS version 2.1 automatically places the destination "all" into the header for each file, unless either allIP or allATM is specified when the file is wrapped. The destination "all" has both ATM and IP queues specified. Therefore, for version 2.1 of GBS most files are sent out over both queues. Figure 4 GBS Gateway Architecture shows the architecture of the Gateway.

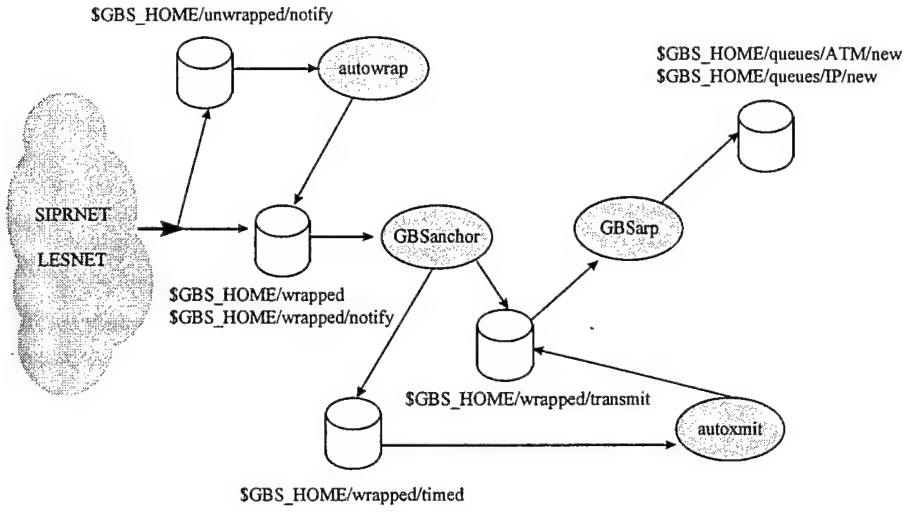


Figure 4 GBS Gateway Architecture

2.3.2 Queue Manager

There is a data file queue for each transport mechanism, ATM and IP. Files are scheduled for inclusion into the broadcast based on values from the wrapper such as the priority and on the internal configuration for each queue. Each queue may be configured as described in the MTN Queue Manager Reference Manual. Figure 5 Queue Manager Architecture shows the queue manager architecture.

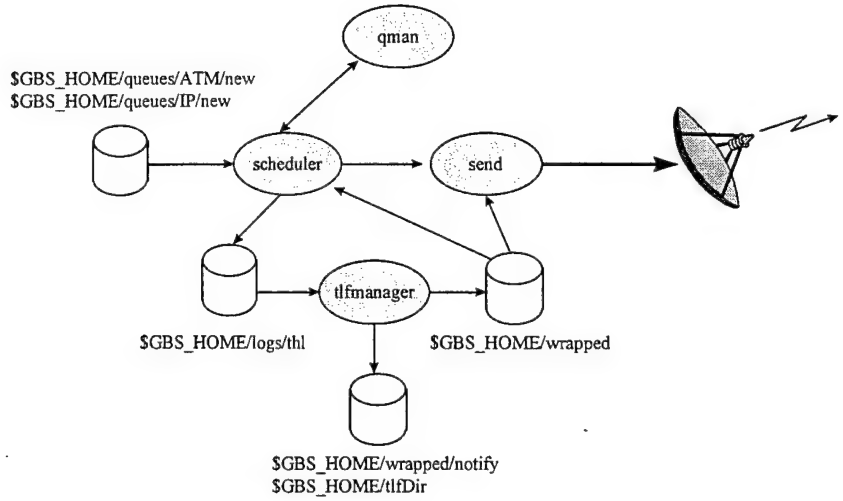


Figure 5 Queue Manager Architecture

When a file is scheduled for transmit, it is added to the broadcast by the *send* program. The *send* program sends data out over UDP port 6566 for the IP transport and over the VP/VC specified in the file $GBS_HOME/config/typeList for that ATM transpor. When a type does not specify a VP/VC, then the VP/VC of the first type within the hierarchy, which specifies a VP/VC, is used. If no VP/VCs are specified for the type or any of its parent types the default VP/VC (0/100) is used.

Periodically the *scheduler* program writes a list of all files that have been sent out over the broadcast into the $GBS_HOME/logs/thl directory. This summary is processed by the *tlfmanager* program. The *tlfmanager* program uses the list along with previous lists that have been generated by the *scheduler* to create a history of summary information on the files that have been sent. Summary information on a file remains in the history for a configurable period of time (6 hours by default). The history of summary information is periodically wrapped using the type `system.thl` and sent over the broadcast so that receive sites can determine if all files sent over the broadcast have been received. The file type `system.thl` should only be used by the *tlfmanager*.

A graphical user interface is provided to allow the user to view the status of the data file queues. This interface is through the program *qman* which is described in the MNT Queue Manager Reference Manual.

2.3.3 Broadcast Status

Data file traffic across the GBS broadcast may not be constant as it is dependent on when the sources insert files into the broadcast. For this reason, the BMC adds a small but constant flow of status information to the broadcast so that the receive sites have feedback on the status of the broadcast at all times. This "heartbeat" is provided through the *statusClient* program which periodically (every five seconds by default) sends small messages over the broadcast on UDP port 6550 for IP and VP/VC 0/90 for ATM. The text of the message is created by BMC operators (initially, the messages are empty strings). This allows the operators of the BMC to notify receive sites of upcoming events, planned outages, etc. A message can be either sent to all queues supported by a Gateway, or a single queue. For queue specific messages, the Queue Manager GUI program *qman* allows the user to enter the textual message to be sent over the

broadcast. A GUI program called *msgGen* allows the user to enter a textual message which will be sent across all queues. The *qman* and *msgGen* programs send the textual messages to the *statusClient* over UDP port 6548; the *statusClient* inserts all status messages into the broadcast for a single queue. The *msgGen* program sends the message to all *statusClient* programs, and the *qman* program sends the message to only the *statusClient* program running on the same machine. Figure 6 Broadcast Status Architecture shows how the status messages are sent to the broadcast.

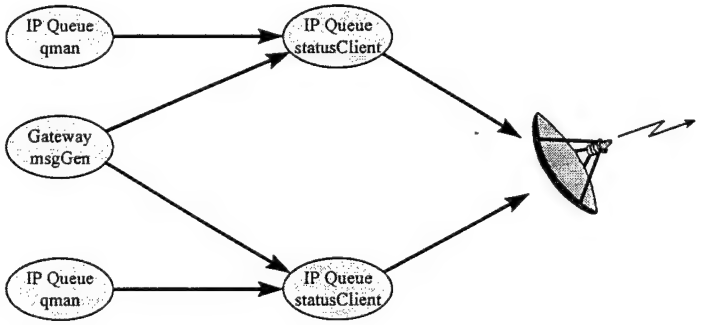


Figure 6 Broadcast Status Architecture

2.3.4 Broadcast Stream Data

Stream data such as TDDS or Binocular can also be sent over the GBS broadcast from the BMC. The stream of information must be received through a serial port in the BMC. The program *streamClient* reads the data from the serial port and adds it to the broadcast. There is no processing of the data; as many bytes as possible are read from the serial port, and then immediately sent over the broadcast. A different copy of *streamClient* is required for each serial data stream to be added to the broadcast and transport mechanism. Therefore, if TDDS data was to be provided to both the ATM and IP broadcasts, two instances of the *streamClient* program would need to be configured and the trap data would need to be received on two separate serial streams. Serial data added to the IP broadcast is sent over UDP ports starting with port 6552. Serial data added to the ATM broadcast is sent over VP/VC's starting with VP/VC 0/91. Figure 7 Broadcast Stream Data Architecture shows the architecture of how streams are broadcast over a single transport.

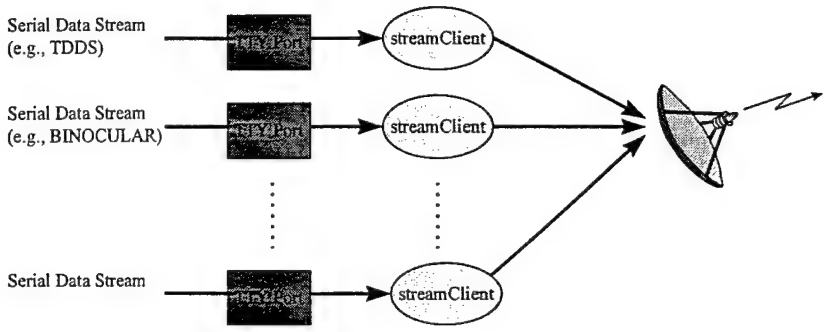


Figure 7 Broadcast Stream Data Architecture

The *streamClient* program is always passed a configuration file as a command line option. This configuration file is first parsed for the generic configuration options and then for stream specific options (the 2 types of options can be interspersed). The pound symbol (#) located in the first column of any line indicates that the line is a comment. Any lines that begin with a space character are ignored during parsing for the generic configuration options (see Appendix C for a detailed list of all options); all stream specific configuration options begin with a space character. The stream programs interpret the debug flag (set through the generic configuration options) to indicate that the data should be duplicated to stdout as well as sent over the broadcast or to a tty port. This is often used to visually verify that data is be transmitted. Table 7 Stream Configuration Parameters describes the different options that are used to configure both the *streamClient* and *streamServer* programs.

Table 7 Stream Configuration Parameters

Parameter	Type	Description	Default Value
IN inputPort	String	tty device to read	/dev/ttya
OUT outputPort	String	tty device to write	/dev/ttyb
S streamName	String	Specifies which stream, used to determine the UDP port or VP/VC	first stream
RS232		This is followed by one or more of the next parameters	
b baud	Integer	Supports baud rates of 300, 600, 1200, 2400, 4800, 9600, 19200, and 38400	9600
s stop_bits	Integer	Must be either 1 (No stop bits) or 2 (stop bit)	No stop bits
pe	none	Even parity	No parity
pE	none	Even parity	No parity

Parameter	Type	Description	Default Value
po	none	Odd parity	No parity
pO	none	Odd parity	No parity
pn	none	No parity	No parity
pN	none	No parity	No parity
d data_bits	Integer	Must be 5, 6, 7, or 8	8 data bits
r rts_cts	Integer		0

Table 8 Sample Stream Configuration File shows a configuration file for a hypothetical news feed data stream. It is important to list all the streams in the same order in each stream configuration file. This list is used to determine which UPD port or VP/VC should be used.

Table 8 Sample Stream Configuration File

```
d 1                     # stream duplicated to stdout (generic configuration option)
s 3                     # Support 3 streams (generic configuration option)
NEWS_FEED               # 1st stream name (following s option)
RADIANT_JADE            # 2nd stream name (following s option)
BINOCULAR               # 3rd stream name (following s option)
 S NEWS_FEED            # (stream specific option)
 RS232 b 2400 pe d7     # (stream specific option)
 OUT /dev/ttya          # (stream specific option)
```

2.3.5 Crypto Synchronization

Another feature provided by the BMC for the IP broadcast is synchronization for the receive cryptos. The *syncerClient* program periodically sends a synchronization data packet (string of zeros) over the IP broadcast UDP port 6551. If the receive crypto is in a state waiting for a synchronization data packet and receives it, it will attempt to regain synchronization. This feature is not required for the ATM broadcast because the ATM switch can provide the resync pulse to the crypto. A corresponding *syncerServer* program is provided for testing and does not run during normal operation.

2.4 GBS Receive

The GBS receive software performs three major functions. The first is to provide feedback on the status of the broadcast to the user. The second is to receive the data files being sent over the broadcast and possibly save them at the receive site. The third is to provide insight into the files that have been received and allow the user to manage these files either from the primary receive workstation or on the local area network. A typical receive site configuration is shown in Figure 8 Receive Suite Architecture.

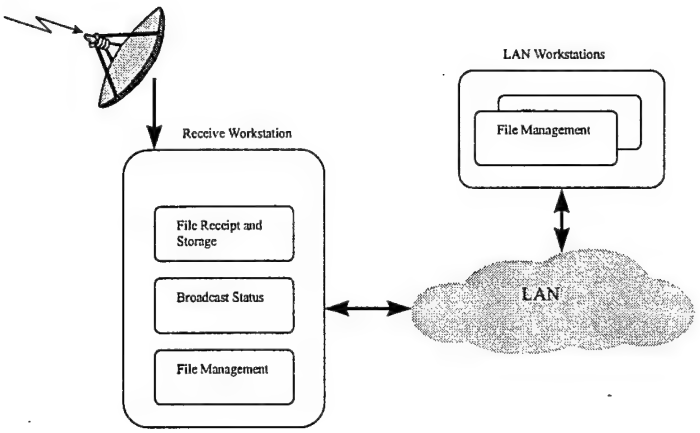


Figure 8 Receive Suite Architecture

2.4.1 File Receipt and Storage

A receive workstation can listen to a single broadcast (ATM or IP). Figure 9 File Receipt and Storage Architecture shows the architecture of the receive workstation.

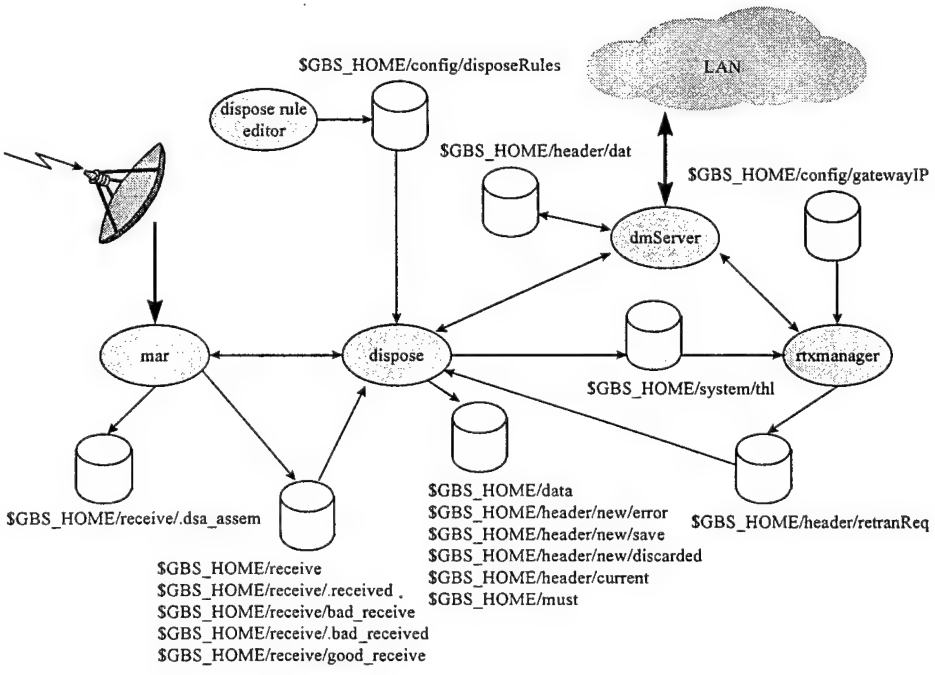


Figure 9 File Receipt and Storage Architecture

The program *mar* reads data from the broadcast on UDP port 6566 for IP transport and from VP/VC 0/100 along with the VP/VC's specified in the file $GBS_HOME/config/typeList for ATM transport. The *mar* program receives information on the size of the file being received as part of the data itself and does not parse the wrapper. The *mar* program reads the file from the broadcast, saving it in $GBS_HOME/receive/.dsa_assem. If the entire file is read successfully, it is moved to the directory $GBS_HOME/receive/good_receive. The .dsa_assem and good_receive directories must be on the same file system so that the move operation is an atomic operation. The *mar* program creates an empty file in $GBS_HOME/receive/.received to indicate that the file has been correctly received.

If there is an error in receiving the file, the file is moved to $GBS_HOME/receive/bad_receive (also on the same file system as the .dsa_assem directory) and the number of bad bytes is written to a file of the same name in $GBS_HOME/receive/.bad_received.

To improve system performance while receiving very large files, *mar* sends a request to the *dispose* process while the file transfer is still in progress. The *dispose* program then reads the wrapper portion of the file and determines whether the file will be saved. Based on the reply from *dispose*, *mar* will either continue saving the file or just discard the remainder of the file.

The program *dispose* determines whether or not each file should be saved on the local file system and the location(s) to store the file. These determinations are made by evaluating the information in the wrapper of the file received against a set of disposition rules. A graphical user interface called the dispose rule editor allows the user to create and edit the disposition rules (the GBS Training and User's Manual describes the dispose rule editor). The first step in determining whether or not to save the file is to collect the set of one or more applicable rules. Which rules are applicable is based entirely on the file's type (specified in the wrapper). All rules whose type exactly matches the received file's type are applicable. If no rules are found, then wildcarded rules of similar type are selected. Of the wildcarded rules that match, only those rules with the most fully specified type are considered for further processing.

In the case where no rules can be found to match the received file's type (with or without wildcarding) then the 'default' rule can be used. There can be only one default rule, which has the type `UNKNOWN`. If this rule exists, then it becomes the only applicable rule. If no rules are found applicable, then the received file is discarded unless the file was wrapped with the must save option. If the file was wrapped with the must save option and no matching rules are found, the file is saved in the directory $GBS_HOME/must.

After the applicable rules have been selected, each of these rules are evaluated against the received file to determine whether they match the file. The rules are evaluated by checking each of the rules criteria to determine whether it matches, If any of the criteria fail, then the entire rule does not match the received file. After the applicable rules are all evaluated, then the received file is saved or discarded based on the matching rules. If no rules matched against the file, then the file is discarded. If any of the matching rules specify that the file should be saved, then it is saved in each of the directories per the rule(s) that matched. A received file can be saved to multiple locations because a single rule can specify more than one save location. In addition, multiple rules can match the same file. If none of the matching rules cause the file to be saved, then the file is discarded. Every file that dispose processes is logged to a system file $GBS_HOME/logs/commandlog which provides a history of what files were discarded or saved along with the reason for the action.

When a file is saved, its date is set to the date on the original file and the file permissions are set according to the wrapper (defaults to the original file's permissions). Also, an expiration time (the time when it will automatically be removed from the system) is determined. The expiration time of each of the rules that caused the file to be saved is examined; the largest of all the expirations times is added to the current time, and this becomes the file's expiration time. If this file is not locked, then it will be automatically deleted after its expiration time.

The directories specified as save locations for the files in the rules are typically under $GBS_HOME/data. However, they may be any directory that is visible from the GBS receive terminal. This includes directories that have been Network File System (NFS) mounted from other machines. Furthermore, the standard

$GBS_HOME/data path may be used in the rules but subdirectories can contain NFS mounted file systems from other machines. In this way, the data received over the GBS broadcast can be made available to other applications on the local area network.

The program *rtxmanager* is the automatic retransmission manager. This program processes files containing summary information on all files that have been sent out from the BMC as described in section 2.3.2. The automatic retransmission manager reads the summary information and queries the data manager to determine if a particular file sent by the BMC has been received. Any files that have not been received are added to an automatic retransmission request. This request is then sent to the BMC by executing a CGI script at the IP address specified in the file $GBS_HOME/config/gatewayIP. If no backchannel exists between the receive site and the BMC the file $GBS_HOME/config/gatewayIP will be empty and no automatic retransmission request will be made. The program *rtxmanager* also checks to determine if the hardware is setup to receive files of the type indicated in the file summary. This check is made by checking the file $GBS_HOME/config/typeList to determine if a multicast address or VP/VC has been assigned to the type. The file $GBS_HOME/config/subscribe.cfg is then checked to determine if the multicast address or VP/VC was being listened to when the file was originally broadcast. Files that have not been received are also added to the data manager so that the user knows that files were sent but not received.

Whenever a new file is processed by either the *dispose* or *rtxmanager* programs, information about the file is saved by sending it to the *dmServer*. The *dmServer* serves as a data repository containing file information about the broadcast files for the receive site. File information about the broadcast files is updated in the *dmServer* by the *dispose* and *rtxmanager* programs and made available via the *dmServer* to other applications on the local area network that provide file management capabilities as described in section 2.4.3.

2.4.2 Receive Broadcast Status

Status information sent out by the Broadcast Management Center (BMC) as described in section 2.3.3 is received and processed by each receive site. The program *statusServer* receives the status information and textual messages sent out by the BMC. Textual messages are saved in the file $GBS_HOME/logs/messagelog. The status information is passed via IPC to the *rdm* program for display to the user. The *rdm* program provides a graphical user interface that shows the status of the broadcast at all times and is described in the GBS Training and User's Manual.

The program *mar* receives the file information from the BMC. This program sends information about the files being received via IPC to the *statusServer* program. The *statusServer* program formats the status information into textual messages and passes these messages to the *rdm* for display to the user. Figure 10 Receive Broadcast Status Architecture shows the architecture of the broadcast status processing at the receive sites.

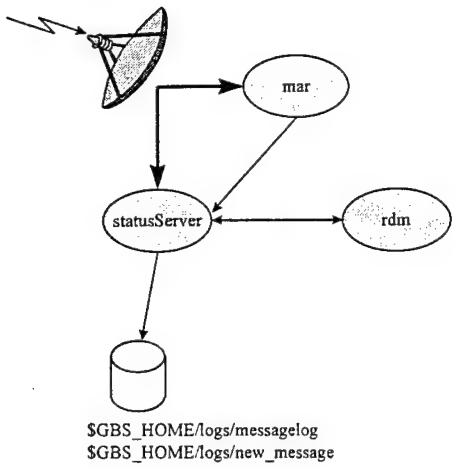


Figure 10 Receive Broadcast Status Architecture

2.4.3 Receive Stream Data

Stream data such as TDDS or Binocular can be received over the GBS broadcast from the BMC. The program *streamServer* reads the data from the broadcast and then writes it to a tty port. There is no processing of the data; as many bytes as possible are read from the broadcast, and then immediately written to the tty port. If the *streamServer*'s configuration file turns on debug, then the data is also written to stdout; this is used to support viewing the data (typically using Oilstock) on the receive workstation. An instance of the *streamServer* program runs on the receive workstation for each stream that the site wants to receive (a receive site does not have to listen to every stream being broadcast). The *streamServer* program is configured in the same manner as the *streamClient* program which is described in section Broadcast Stream Data. Figure 11 Receive Stream Data Architecture shows how the architecture supports receiving streams.

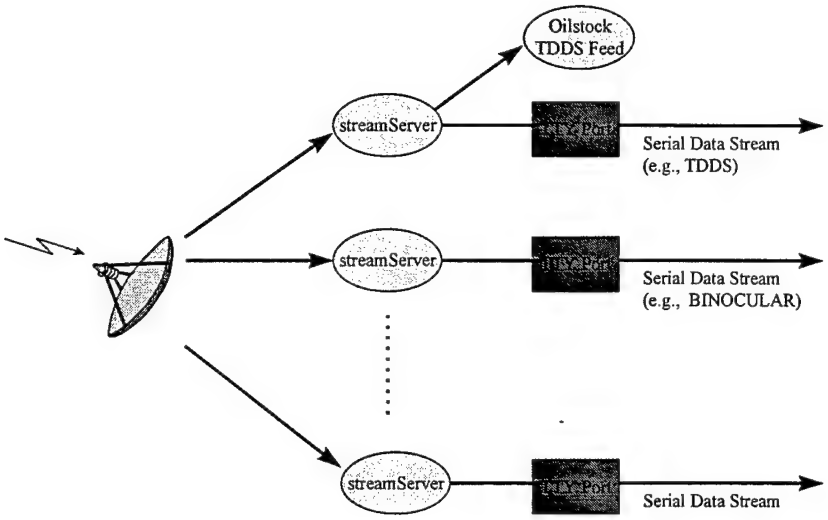


Figure 11 Receive Stream Data Architecture

2.4.4 File Management

A list of all files known by the receive site is saved to provide insight into the workings of the broadcast. More importantly, it provides the user easy access to information on what has been sent over the broadcast, what has been received, and where the files that were received are saved. The program *dmServer* maintains the list of information on all files known by the receive site. This program is known as the Data Manager and maintains the list of file information in a flat file of fixed length records in the directory $GBS_HOME/header/dat.

Other information such as the type list from the configuration file $GBS_HOME/config/typeList is also stored in fixed sized flat files containing fixed length records within the directory $GBS_HOME/header/dat. These additional files allow the *dmServer* to create simple cross references to data such as the file type which greatly reduces the amount of space required to store the type of each file. The files in the directory $GBS_HOME/header/dat are strictly for use by the program *dmServer* and should NOT be changed by any other means. To provide the best possible performance, the *dmServer* program maps the flat files into memory, so they must be stored on a local file system on the machine where the *dmServer* runs. The data manager server is currently sized to support information on 100,000 broadcast files. Table 9 Data Manager Data Files describes the four data files used by the *dmServer*.

Table 9 Data Manager Data Files

Data File	Description	Max Records	File Size
typeMap	Contains the list of known file types with an identifier for each (similar to relational database reference table).	500	44 KB
markingMap	Contains the list of known file markings with an identifier for each (similar to relational database reference table).	200	16 KB
fileList	Contains the list of files, including saved, discarded, deleted files. This contains a subset of the meta data from the wrapper which can be viewed through the GUI file managers.	100000	27.2 MB
dispositions	Contains the directories where any currently saved files are stored.	150000	13.8 MB

The *dmServer* program provides an Application Program Interface (API) that allows multiple programs to connect to the server and make requests. The types of requests supported include: retrieve the reference data (types and markings), search for a list of files, add file records, and lock/unlock/delete files. This interface also supports asynchronous notification of events such as new files, lock/unlock files, delete files. The *dmServer* normally runs on the receive workstation, although this is not necessary; it can run on any networked Sun workstation.

A test program, *dmTestAPI*, is provided to debug problems and verify the data file integrity. This program is provided for convenience only; it is not necessary for the normal operation of a receive site. This program displays a row ID with each data record; this row ID provides the same information as a relational database key. The row IDs are used internally, and never presented to the user by either *rfm* or *nfm*. The *dmTestAPI* program is a menu driven command line program. When *dmTestAPI* is started, it presents the user with the Connection Menu. Table 10 dmTestAPI Menus describes the menu options.

Table 10 dmTestAPI Menus

Connection Menu	
Menu Item	**Description**
Connect to Data Manager	Connect to the data files or *dmServer* based on the current setting of the connection mode. After the connection is made, the Request Menu will be presented.
Change to Data Manager server mode	Current state is set to access the data files directly, not through the *dmServer*. Selecting this toggles the connection mode. This is the default setting.
Change to Data Manager test mode	Current state is set to access make requests through the *dmServer*. Selecting this toggles the connection mode. The request options with the comment "(not in API)" are not available when the *dmTestAPI* is making requests through the *dmServer*.
Exit	Exit the program
Request Menu	
Menu Item	**Description**
Print This List	Redisplay the list of menu options
Get Types	Retrieves the type reference table and prints the new list
Get Markings	Retrieves the marking reference table and prints the new list
Add New Type	Prompts to add a new file type

Add New Marking	Prompts to add a new file marking
Print current Type Map	Prints the *dmTestAPI*'s cached type map
Print current Marking Map	Prints the *dmTestAPI*'s cached marking map
Generate Test Records	Prompts for information to add new file records
Lock File	Prompts for a file row ID to lock
Un-Lock File	Prompts for a file row ID to unlock
Mark File for Delete	Prompts for a file row ID to 'delete'. This actually just marks the file for deletion, it won't be deleted until cleanup runs. However, this file will not appear in saved file query requests
Retrieve Dispositions	Prompts for a file row ID to retrieve saved file locations
Delete file dispositions (not in API)	Not currently supported
Free file (not in API)	Prompts for a file row ID to remove from the data files.
Last 10 Active Rows	Retrieves the 10 most recent rows (not limited to saved files)
1st 10 Active Rows	Retrieves the 10 oldest rows (not limited to saved files)
Search	Prompts for search criteria. This search supports the entire set of criteria that the *dmServer* supports.
List all file details	Toggles an internal flag to the *dmTestAPI*. When file rows are printed, all the information stored for the row is displayed.
List file with summary information	Toggles an internal flag to the *dmTestAPI*. When file rows are printed, only a subset of the information stored for the row is displayed. This is the default setting.
Mask out Row ID	Toggles an internal flag to the *dmTestAPI*. When file rows are displayed, always show the row ID as 0. This supports testing the compression algorithms which change the row IDs.
Print correct Row ID	Toggles an internal flag to the *dmTestAPI*. When file rows are displayed, always show the correct row ID. This is the default setting.
Print All Rows (not in API)	Retrieves all active records (in time order). This can be used to dump the contents of the data files to an ascii file.
Data Manager Status	Retrieves statistics about capacity of the data files.
Count Active Rows (not in API)	Walks through the entire file list to count the active records; this supports testing the data file integrity.
CleanUp - Expire Files	Causes all files whose expiration time has passed and are not locked to be marked for delete. Changes the status of any files that are marked for delete to deleted, and removes any saved files that no longer have a file record with a status of saved. This function is available through *dmUtility* -expire.
CleanUp - Purge Deleted Files	Remove file records that are older than the specified threshold and do not reference saved files. This function is available through *dmUtility* -purge <olderThanHours>.
Verify Data Files	Verifies the data file integrity by performing many different checks. This will prompt whether it should automatically attempt to fix problems (most cannot be automatically fixed at this time). Some problems will be fixed even if you do not request to fix problems. This function is available through *dmUtility* -verify.
Compact Data Files	Resequences the fileList data file to provide optimum access. This function is available through *dmUtility* -compact.

Expand Data Files	Increases the size of the data files. This function is available through *dmUtility* -expand.
Disconnect from Data Manager	Disconnect from the Data Manager. After the connection is broken, the Connection Menu will be presented.
Exit	Exit the program

The program *dmConvert* is used to initially create the data files. If the data files become corrupted, then *dmConvert* can be run again to re-create the data files from the header files that have been saved.

The *dmUtility* is a command line program that calls certain *dmServer* API calls based on command line arguments. This is automatically run periodically to perform maintenance on the data files including expiring, purging, and compacting as described in Table 10 dmTestAPI Menus.

Figure 12 File Management Architecture shows the architecture that allows information saved in the data manager to be accessed. Information on files received over the broadcast and on files that were sent over the broadcast are created in the data manager as described in section 2.4.1. These entries can be viewed through one of two graphical user interface programs. The first program called *rfm* (receive file manager) is a Motif based application that will only run on UNIX systems. The second program called *nfm* (Netscape file manager) is a web based application that will run on any machine with a web browser and access to the machine on which the *dmServer* program is running. Both *rfm* and *nfm* are described in the GBS Training and User's Manual. Both of the file management user interface programs interact with the Data Manager to obtain information on the files that have been sent and possibly received over the broadcast. Furthermore, these programs can be run on any machine that is networked to the machine on which the *dmServer* is running.

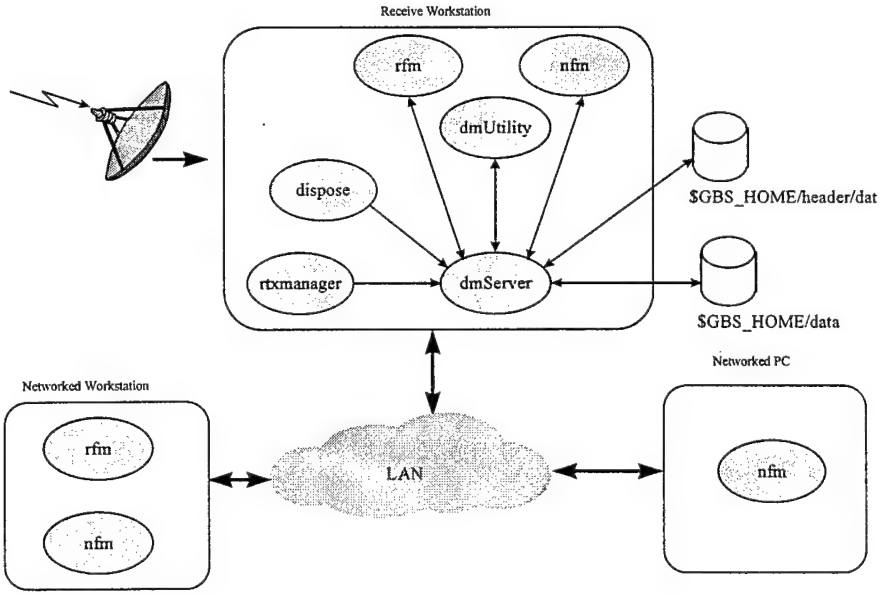


Figure 12 File Management Architecture

2.4.5 Executive

The *executive* program provides a window across the top of the screen containing a clock depicting the time in Zulu format, and a pane of buttons. The button pane displays a menu when one of the buttons is selected, which is used to activate various functions or launch programs. The window geometry can be completely controlled via entries in the *executive*'s configuration file. The *executive* supports interprocess communication via optional pipes to a program's stdin, stdout, and stderr, as well as via sending UNIX signals to application programs.

The *executive* uses a Resource File, $GBS_HOME/app-defaults/Executive, to define some of the appearance of its widgets. Two features that can be implemented in the resource file are Menu mnemonics and accelerators.

The *executive* redirects stdout from programs that it starts. The messages are sent to the system console: either the *executive*'s console if enabled, or the standard UNIX console window. The *executive* reads a configuration file, $GBS_HOME/config/gbs.config, to determine the layout of the pane of buttons and associated menus. See Appendix D Executive Configuration File for details about the format and content of this file.

Appendix A GBS Programs

This appendix lists all of the GBS programs and gives a brief discussion of what they do.

Program Name	Type	Location	Description
GBSanchor	daemon	Gateway	Determines whether a file is immediately forwarded to *GBSarp* for broadcast or should be sent to *autoxmit* for timed transmit based on the wrapper. Most files are immediately forwarded for broadcast.
GBSarp	daemon	Gateway	Determines which transport(s) each file should be sent over based on the destinations specified in the wrapper.
GBSblert	GUI	Receive	Provides statistics on the Block Errors in the broadcast.
GBSstats	GUI	Receive	Provides network statistics, supporting both IP and ATM. Currently not being used
Qmgr	GUI/daemon	Q Mgr	Original Queue Manager, this program needed to be running for files to be broadcast. No longer used
QmgrATM	GUI/daemon	Q Mgr	Symbolic link to Qmgr, indicates that this program instance is managing the ATM queue. No longer used
QmgrIP	GUI/daemon	Q Mgr	Symbolic link to Qmgr, indicates that this program instance is managing the IP queue. No longer used
View	GUI	All	Provides an X Windows based interface to view log files.
alarm_manager	daemon/GUI	Receive	Started by the *executive* to display an alarm window to the user when an alarm condition exists. The alarm events are generated by other programs (*devicechk*, *heartmonitor*, etc.) which create an alert file in the /tmp directory.
alarm_popup	GUI	Receive	Supports the *executive* with alarm notification Not used any longer(?)
applinker	GUI	Receive	Allows users to associate file extensions/types with applications for launching by the GBS file managers *rfm* and *nfm*.
audio_alert	daemon	Receive	Periodically sends 3 beeps to the system console if the current broadcast status is down and audio alarms are not disabled.
autowrap	daemon	Gateway	Looks for notify files in the $GBS_HOME/unwrapped directory and wraps the corresponding file.
autoxmit	daemon	Gateway	Supports timed transmit of files by periodically touching the notify file for *GBSarp* based on the parameters in the wrapper.
autoxmitUI	GUI	Gateway	Displays which files are being periodically retransmitted and allows the operator to manage these files.
chdate	command line	Utility	Changes a file's timestamp. This utility was written to support maintenance shell scripts (this functionality could possibly be provided using the UNIX touch -t command).

Program Name	Type	Location	Description
default_gen	command line	Source	Creates a defaults file for wrapping based on command line arguments.
devicechk	daemon	Receive	Periodically checks disk capacity against user specified criteria (see *devicegui*) and generate alarm files in the /tmp directory if an alarm condition exists.
devicegui	GUI	Receive	Allows the operator to specify which file systems to monitor for capacity and at what threshold to generate an alarm.
dispose	daemon	Receive	Determines whether files received over the broadcast should be saved based on the dispose rule criteria (see *dre*).
dmConvert	command line	Receive	Creates the data files used by *dmServer*.
dmServer	daemon	Receive	Manages the list of files known on a receive system. Provides an interface for other programs to submit new data as well as query the current data.
dmTestAPI	command line	Receive	Test support program, interfaces with the *dmServer*.
dmUtility	command line	Receive	Utility program to make periodic requests to the *dmServer*.
dre	GUI	Receive	Allows the operator to edit the dispose rules.
execabout	GUI	Receive	Displays an 'About' screen for the *exectutive*. Due to the way the *executive* menu is configured, it is much simpler to make this a standalone program.
exechelp	GUI	Receive	Displays the help screen for the *exectutive*. Due to the way the *executive* menu is configured, it is much simpler to make this a standalone program.
executive	GUI	Receive	Provides a system menu bar which is highly configurable.
filehtml	command line	Receive	Supports the *nfm* by formating the HTML page for the file frame.
filewrap	GUI	Source	Allows the operator to edit a wrapper. This is normally started by *wrap*.
filterFlash	GUI	Receive	Generats a popup when started by *dispose* because a FLASH or FLASH IMMEDIATE message was received. No longer used
findlogs			
gbstp_recvATM	daemon	Receive	Original ATM transport program to receive files. No longer used
gbstp_recvIP	daemon	Receive	Original IP transport program to receive files. No longer used
gbstp_sendATM	daemon	Q Mgr	Original ATM transport program to broadcast files, started by Qmgr. No longer used
gbstp_sendIP	daemon	Q Mgr	Original IP transport program to broadcast files, started by Qmgr. No longer used
heartmonitor	daemon	Receive	Started by the *executive* to monitor the broadcast status file, producing an alarm if the broadcast is lost.
logroll	GUI	Receive	No longer used

Program Name	Type	Location	Description
mar	daemon	Receive	Transport program to receive files from the broadcast.
messagemonitor	daemon	Receive	Generates alarm files in the /tmp directory when started by *dispose* because a FLASH or FLASH IMMEDIATE message was received.
monitor			
msgGen	GUI	Gateway	Allows the operator to specify a BMC status message that is sent over all broadcast queues supported by the gateway.
pmDriver	command line	Utility	Provides an interactive command line interface to queue the *processMonitor* about the status of processes being managed.
popup	GUI	Utility	Creates a window containing a text message based on command line arguments. Not currently used
printwrap	command line	All	Reads a wrapper and formats the data into a text summary suitable for presentation to users.
procclient	GUI	Receive	Interacts with the *processMonitor* to allow the operator to view the status of all managed programs.
processMonitor	daemon	All	Manages processes based on configuration files, supports continuous, periodic, and timed processes.
prodtypegen	command line	Q mgr	Parses typeList file to extract multicast and VP/VC settings which are written to the prodtype.defs config file.
pte	GUI	Q mgr	Allows the operator to edit the product type database which affects the scheduling of file transfers based on file type.
qman	GUI	Q mgr	Provides a status of the current file queue activity and allows the operator to manage the queues.
rdm	GUI	Receive	Provides a broadcast status to operators including indication of whether broadcast is up or down, list of files currently be received, and status messages.
reset_dtr	command line	Receive	Initiates script to resync the crypto by toggling the dtr. This is automatically started by the *statusServer* when the broadcast is lost for 3 consecutive status message time-outs.
resyncparse	command line	Receive	Processes the broadcast status log to generate statistics about periods of broadcast outages.
rfm	GUI	Receive	Receive file manager that allows an operator to query for files and manage those files.
rtxmanager	daemon	Receive	Processes Transmission History Log (THL) files and requests re-transmission of files that were not received.
runcmd	GUI	Receive	Provides an X Windows user interface to run command line programs (specified on the command line).
rxt	GUI	Q mgr	Allows the operator to edit the receive terminal database which affects the scheduling of file transfers based on when receive sites are available.
scheduler	daemon	Q mgr	Manages the file queues and initiates the transfer of all files based on priority, product type and receive terminal database settings, and scheduling algorithms.
send	daemon	Q mgr	Transport program to send files to the broadcast.

Program Name	Type	Location	Description
startNetscape	command line	Utility	Launches Netscape with the URL specified on the command line. If Netscape is already running, then the existing Netscape will be asked to display the page instead a launching another Netscape.
statusClient	daemon	Q mgr	Sends status messages over the broadcast, selects either IP or ATM protocol based on command line argument.
statusServer	daemon	Q mgr	Receives status messages from the broadcast, selects either IP or ATM protocol based on command line argument.
streamClient	daemon	Receive	Sends a stream of data over the broadcast, selects either IP or ATM protocol based on command line argument.
streamServer	daemon	Receive	Receives a stream of data from the broadcast, selects either IP or ATM protocol based on command line argument.
subexec	GUI	Receive	Supports the *executive* program.
subscribeMgr	daemon	Receive	Waits for a SIGUSR1 signal indicating that the typeFile has been modified. Updates the subscribe.cmd and subscribe.cfg files and then sends a SIGUSR1 signal to *mar*.
switchQM	command line	Q mgr	Switches between old and MTN Queue Managers / transports. No longer used
switchRX	command line	Receive	Switches between old and MTN transports. No longer used
syncerClient	daemon	Q mgr	Sends sync packets over the broadcast.
syncerServer	daemon	Receive	Receives sync packets from the broadcast. Not currently used
tlfmanager	daemon	Q mgr	Parses the transmission log to periodically create and send a Transmission History Log (THL) file.
typehtml	command line	Receive	Supports the *nfm* by formating the HTML page for the type frame.
wrap	GUI	Source	Allows operators to wrap files.
wrapit	command line	Source	Command line interface to wrap files.

Appendix B GBS Shell Scripts

This appendix lists all of the GBS shell scripts and gives a brief discussion of what they do.

Script Name	Description
Kill_naudio	Checks for all Paradise audio software running and kills it.
Kill_video	Checks for all Paradise video software running and kills it.
QmgrATM.startup	Startup script for old Queue Manager. No longer used
QmgrIP.startup	Startup script for old Queue Manager. No longer used
RDM.startup	Startup script for *rdm* program to initialize the environment.
Start_Video	Starts Paradise video software
addGBSuser	Adds a new user (updating UNIX system files) and configures the user to access GBS. Can also be used to convert an existing user's account to access GBS.
addprinters	Configures a system (updating UNIX system files) to use a printer.
archiveManager	Automatically run periodically to archive and truncate log files.
audiorate	Calculates statistics (about Paradise audio quality?)
autoretranstype	Supports configuration of what files should be automatically requested for retransmission.
cleanupReceive	Automatically run periodically to remove old processing files and archive log files.
convert_rdmconfig	Utility used during GBS software upgrade (version 1.x to version 2.1x) to convert an existing dispose rule file (rdmconfig) to the new format (dispose.cfg).
gbs_links	Makes symbolic links to the dot files (.cshrc, .login, .openwin-init, etc) in the gbs user account in the current directory. Not currently used
html_setup.pl	Initializes the environment for web based file wrapping Not currently used
initializeDSAQueueManager	Runs *prodtypegen* program to ensure that *send* has the correct multicast and VP/VC addresses initially.
load_oil_map	Initializes new Oilstock maps.
logManager	Automatically run periodically on source and gateway systems to remove old wrapped files.
msgGen.startup	Startup script for the *msgGen* program to initialize the environment.
naudio.startup	Starts Paradise audio software.
process_retx.cmd	Supports web based retransmission requests.
pulse_dtr.exp	Expect script to resync the crypto by toggling dtr. The file contains commands (with expected responses) to telnet into the router and toggle the dtr line.
removeBigFiles	Utility to help operators find large files that can be removed when they are running out of disk space. No longer used
runDSAQueueManager	Determines whether the system is configured for the MTN queue manager.
runOSOQueueManager	Determines whether the system is configured for the old OSO queue manager.
systemShutdown	Prompts the operator with instructions about how to shut the system down safely.

Script Name	Description
update_vc	Signals programs because the typeList has changed and the current VP/VCs may not be correct. No longer used
wrap.startup	Startup script for the *wrap* program to initialize the environment.
xferit	Daemon process to automatically forward wrapped files on a source machine to the gateway.

Appendix C Generic Configuration Parameters

Most GBS programs (with the exception of MTN and *executive* related programs) parse a generic configuration file every time they are started; this file is $GBS_HOME/config/default.config. This appendix lists all options currently supported in this configuration file.

While the file is being parsed, a pound symbol (#) located in the first column of any line indicates that the line is a comment. Any lines that begin with a space character are ignored during parsing as well as any lines that do not begin with one of the options specified below (no error messages are generated).

Parameter	Type	Description	Default Value
a atmDevice	String	Name of the ATM device	/dev/fa0
B numBuckets maxSize maxSize ...	Integer Integer	No Longer Used Define buckets for transfers. First line indicates the number of buckets not including the flash bucket <= 6. The line beginning with the B option must be followed by <numBuckets> lines, each containing the max size of transfers on those buckets. Numbers should be in ascending order. Each bucket goes from prev value to max value. The last bucket must have max size 0.	2 0 50
b baseDir	String	Path to GBS directory	$GBS_HOME if set, otherwise /home/GBS
c numServers	Integer	Maximum number of servers running numServers <= 8	1
C Server	Integer	Which server instance < 8	0
d debug	Integer	Enables debugging information to be printed to the errorlog file in the logs directory. 0: all debug output is disabled 1: standard debug output 2: standard and library debug	0
D dataManagerHost	String	Host name where the Data Manager is running	"GBS-DataManager"
f thltimeframe	Integer	No Longer Used	300
h hold_queues	Integer	No Longer Used Hold queues when the queue manager starts	0
I thlinterval	Integer	No Longer Used	60
i cnt	Integer	No Longer Used	100
j cnt1	Integer	No Longer Used	100
l tickLength	Integer	No Longer Used	131072
L dmLockDownMmapPages	Integer	If this is non-zero, then the Data Manager will attempt to lock the file information data file into physical memory.	0
N myName	String	? Should only be used during testing.	Machine's nodename

Parameter	Type	Description	Default Value
o ServerOffset	Integer	This <ServerOffset> is multiplied by <Server (from the C option)> and added to the VP/VC specified in the typeList file to determine which VP/VC to send a file over on the ATM transport.	50
PA statusPort	Integer	UDP port number for status messages	6550
PC commPort	Integer		6549
PF gbstpPort	Integer	No Longer Used ??? First UDP port number for file data	6560
PM messagePort	Integer		6548
PS streamPort	Integer	First UDP port number for stream data	6552
PZ syncerPort	Integer	UDP port number for sync packets	6551
q highQual mediumQual lowQual	Integer Integer Integer	No Longer Used Define the number of transfers for the quality level specified by the wrapper	1 1 1
Q numQs Qname Qname ...	Integer String	No Longer Used Define queues for transfers. First line indicates the number of queues < 10. The line beginning with the Q option must be followed by <numQs + 1> lines, each containing the name of the queues. Queues should be in order of priority. Each queue name can be no more than 20 characters. The first queue is the override queue. To not use this, make the first line blank	4 "FLASH OVERRIDE" "FLASH" "IMMEDIATE" "PRIORITY" "ROUTINE"
r nfmRefresh	Integer	No Longer Used Number of seconds between when the Netscape File Manager queries the Data Manager for the latest files received.	30
S TimeSliceLength numSlices numSlices ...	Integer Integer	No Longer Used Define slices for flow control and time sharing of the buckets. First line indicates the length of the time slice (in milliseconds). The line beginning with the S option must be followed by <numBuckets (from the B option)> lines, each containing the number of slices allocated to that queue in each round robin.	1 100 2 2
s numStreams streamName streamName ...	Integer String	Defines the data streams. First line specifies the number of streams which must be <= 8. The line beginning with the s option must be followed by <numStreams> lines, each containing the name of the stream.	2 "TRAP" "TIBS"
t syncTime	Integer	The number of seconds between sync packet transmissions	10

Parameter	Type	Description	Default Value
T statusTime	Integer	The number of seconds between status message transmissions	5
u	None	Sets super user status to True (used during wrapping)	False
VA status_vp/status_vc	Int / Int	ATM VP/VC for status messages	0/90
VF default_vp/default_vc	Int / Int	No Longer Used First ATM VP/VC for file transfers	0/100
VS stream_vp/stream_vc	Int / Int	First ATM VP/VC for stream data	0/91
x numTransmissions	Integer	No Longer Used	1
X numTransports transportName transportName ...	Integer String	No Longer Used Define transports for file transfers. First line indicates the number of transports. The line beginning with the X option must be followed by <numTransports> lines, each containing the name of the transport. Each transport name can be no more than 10 characters.	3 "IP" "ATM" "FBS"
X syncLength	Integer	The number of zeros sent in each sync packet	50

Appendix D Executive Configuration File

The *executive* configuration file, $GBS_HOME/config/gbs.config, defines the *executive* window layout, the application processes, and the menus used to control these processes. Each line starts with a keyword. Most keywords are then followed by one or more values. Comments may be added to the end of any line, following a '#' character. There may be blank lines in the file, and comment lines starting with '#'. A '#' character may be preceded by a '\' (forward slash) if it should not be interpreted as a comment indicator. To put a newline in a Motif label specifier, simply use "\n". Any keyword for which the default value is acceptable, or which does not apply, may be omitted. Environment variables will be translated anywhere they appear in the configuration file. To reference an environment variable put ${ENV_VRBL_NAME} in the configuration file. This may be preceded by a '\' (forward slash) if it should not be interpreted as an environment variable reference.

Each keyword name below includes the keyword name, the type of values required (if any), the units of the value (inside parens), and any range limits on the value(s) (inside square brackets). The type of value is one of:

- integer (just an integer number)
- string (a string containing no white space)
- string_with_blanks (a string which may contain whitespace - the string is considered to be all characters between the first non-whitespace character and the last non-whitespace character before the '#' or the newline - if blanks are desired at the beginning or end of the string enclose the string in double quotes).

Some keywords include a description of which keyword block(s) it may appear in. Keyword blocks begin with a *_DEFN keyword, and continue until the next *_DEFN keyword. If this line doesn't appear, then the keyword may only appear within a block starting with the same prefix (e.g MI_LABEL must appear in an MI_DEFN block).

General Executive Parameters

These keywords may appear anywhere within the configuration file, but it is best to put them all at the beginning for readability.

Keyword	Description
X_MAX_PROCS integer [1 <= value <= 30] Default: 30	The maximum number of processes (not counting the main *executive* process), that may be running at once.
X_SPAWN_DELTA integer (seconds) [>=0] Default: 5	Minimum interval in which processes will be spawned. This is provided to prevent the user from spawning lots of processes real fast and bringing the system to its knees. A value of 0 disables this delay.
X_EXITMSG_COUNT integer [>=0] Default: 0	When the *executive* must exit due to a catastrophic error, all windows will disappear. So, in order for the user to see the exit message, it can be displayed several times (in either the logger window or the console window) in order to catch his attention, before everything goes blank. This is the number of times the message will be displayed. Or, if this value is 0, then the message will be displayed in an OK Popup (once).
X_EXITMSG_DELTA integer (seconds) [>=1] Default: 2	See remarks for X_EXITMSG_COUNT. This is the delay in seconds between displaying the message. This is ignored if EXIT_MSG_COUNT is 0.

Keyword	Description
X_LOGOUT_DELAY integer (seconds) [>=0] Default: 5	Delay in seconds after sending SIGTERMs to all running processes before SIGKILLs are sent to all processes that are still running and the *executive* exits. A value of 0 disables the SIGTERMs; the *executive* just sends SIGKILLs to all running processes and exits.
X_PROCNAME_CHARS integer [1 <= Value < 19] Default: 7	The number of characters in the process name field for log messages. The field will always be this wide, so successive log messages line up nicely. Process names will either be truncated or blank filled as required to yield this many characters.
X_EXECNAME string [non-null, <20] Default: "exec "	The name of the main *executive* process to use in log messages. It will be truncated if necessary.
X_EXEC_EXEC_KEY string [non-null, <10] Default: "!E"	Keystring used to identify *executive* to sub-executive messages. If a sub-executive will be created, this keystring MUST be set to something with which no application message will ever start.
X_PROC_EXEC_KEY string [non-null, <10] Default: Capability disabled	Application processes whose stdout is piped to the *executive* may send messages starting with this keystring to the *executive*. The message consists of this keystring, a blank, an action set tag, and a newline. The *executive* will execute the action set when it receives this message. This provides application processes a mechanism to operate on *executive* menu items and to request the *executive* to spawn processes.
X_DEBUG_FILE string [DBG_SMRY, DBG_MENU, DBG_MSGS, DBG_RUN] string ["stdout", or the pathname of a file [non-null, <80]] Multiple: OK as long as DBG_* is different on each occurrence Default: capability disabled	Each DBG_ keyword defines a category of *executive* debug print. Each category can be sent to the *executive*'s stdout or to a file. If the same pathname appears for more than one category, the debug print for those categories will go to the same file. It is a good idea to put this at the beginning of the config file, since a debug flag may enable debug related to parsing other keywords in the config file itself.

Executive Window Layout Parameters

By default, only the frame and button panel are created.

Keyword	Description
W_FRAME Default: Frame is always created if at least one of the Console, Clock, or Buttons subwindows is created	Specifies the start of a block of window definition parameters for the frame, consisting of some or all of these keywords: W_FRAME_LABEL, W_LEFT, W_TOP, W_WIDTH, W_HEIGHT, W_FONT
W_CONSOLE Default: Console subwindow is not created	Indicates that a console subwindow should be created. Also may specify the start of a block of window definition parameters for the console, consisting of some or all of these keywords: W_LEFT, W_TOP, W_WIDTH, W_HEIGHT, W_FONT
W_BUTTONS Default: Buttons subwindow is created if there are any M_DEFN blocks	Specifies the start of a block of window definition parameters for the frame, consisting of some or all of these keywords: W_FRAME_LABEL, W_LEFT, W_TOP, W_WIDTH, W_HEIGHT, W_FONT

Keyword	Description
W_CLOCK Default: Clock subwindow is not created	Indicates that a clock subwindow should be created. Also may specify the start of a block of window definition parameters for the clock, consisting of some or all of these keywords: W_LEFT, W_TOP, W_WIDTH, W_HEIGHT, W_TIME_FONT, W_DATE_FONT, W_GMT_FONT
Background on W_LEFT and W_TOP	Some of the following may no longer apply. See the remarks under W_POSITION about using W_LEFT and W_TOP sparingly. A value >= 0 is the position in pixels. A value of -1 means do normal default positioning, using "right of else below" rules. A value of -2 means use special positioning rules (only valid for W_LEFT with W_CLOCK and W_TOP with W_BUTTONS).
W_LEFT integer (pixels) [>=-2 for W_CLOCK, else >=-1] Default(W_FRAME): 11 Default(W_CONSOLE, W_BUTTONS): 0 Default(W_CLOCK): -2 which means right of console if it is enabled, else right of buttons	The x coordinate of the left edge of the window. -1 means position using default rules. -2 for W_CLOCK means position right of console if enabled, else right of buttons. Valid within blocks: W_FRAME, W_CONSOLE, W_BUTTONS, W_CLOCK
W_TOP integer (pixels) [>=-2 for W_BUTTONS, else >=-1] Default(W_FRAME): 27 Default(W_CONSOLE, W_CLOCK): 0 Default(W_BUTTONS): -2 which means below console if it is enabled, else use default position	The y coordinate of the top edge of the window. -1 means position using default rules. -2 for W_BUTTONS means position below console if enabled, else use default rules. Valid within blocks: W_FRAME, W_CONSOLE, W_BUTTONS, W_CLOCK
Background on W_WIDTH and W_HEIGHT	Some of the following may no longer apply. See the remarks under W_POSITION about using W_WIDTH and W_HEIGHT sparingly. A value of -2 (only for the clock) means make same height as console if it exists, else 64 pixels high. A value of -1 means shrink in that dimension to fit contents (except for the console for which this doesn't make sense). A value of 0 means expand to right or down to reach to the edge of the frame (or in the case of the frame expand to the edge of the screen). This only works if that dimension of the frame is specified explicitly in pixels, or if it is "extend to edge of screen". A positive value is the dimension in pixels, except for the console in which case it is the dimension in characters.

Keyword	Description
W_WIDTH integer [>=1 for W_CONSOLE, else >=-1] Default(W_FRAME, W_BUTTONS, W_CLOCK): 0 Default(W_CONSOLE): 107 (Xview) or 110 (Motif)	The width of the window in pixels (or characters for W_CONSOLE). A value of 0 means expand to the right edge of the window (or screen in the case of W_FRAME) containing this window. A value of -1 means shrink to fit contents (not available for W_CONSOLE). Valid within blocks: W_FRAME, W_CONSOLE, W_BUTTONS, W_CLOCK
W_HEIGHT integer [>=1 for W_CONSOLE, >=-2 for W_CLOCK, else >=-1] Default(W_FRAME, W_BUTTONS): -1 which means fit to the window's contents Default(W_CONSOLE): 6 Default(W_CLOCK): -2	The height of the window in pixels (or characters for W_CONSOLE). A value of 0 means expand to the bottom edge of the window containing this window. A value of -1 means shrink to fit contents (not available for W_CONSOLE). A value of -2 (only available for W_CLOCK) means special rules: same height as console if it exists, else 64 pixels high. Valid within blocks: W_FRAME, W_CONSOLE, W_BUTTONS, W_CLOCK
Background on W_POSITION	The *executive*'s console, clock, and buttons subwindows are placed inside a Motif Form widget, which makes it possible to handle window fitting and resizing nicely. By using the W_POSITION keyword you can specify the relative locations of these subwindows. If you are creating all 3 windows, then you must specify one window on an edge (e.g. "BOTTOM"), and the other two windows in the opposite corners (e.g. "TOP_LEFT" and "TOP_RIGHT"). If you are creating only 2 windows, one of them must specify an edge. If you are creating only 1 window, there is nothing to do because it will fill the frame anyway.
W_POSITION string ["TOP", "BOTTOM", "LEFT", "RIGHT", "TOP_LEFT", "TOP_RIGHT", "BOTTOM_LEFT", "BOTTOM_RIGHT"] Default(W_CONSOLE): "TOP_LEFT" Default(W_CLOCK): "TOP_RIGHT" Default(W_BUTTONS): "BOTTOM"	Be careful to select a consistent set of choices for the subwindows you will be creating, or else you may get a very strange window geometry. In general, you should use this keyword to layout your subwindows, and use the W_LEFT, W_TOP, W_HEIGHT, W_WIDTH keywords only when necessary. Valid within blocks: W_CONSOLE, W_BUTTONS, W_CLOCK
W_FONT string [non-null, < 80) chars] Default: Use default font	The name of a font to use for the window's font. A fixed width font is strongly recommended for the console. Use the 'xlsfonts' command to find a list of available fonts. Valid within Motif blocks: W_CONSOLE, W_BUTTONS (Motif only)
W_FRAME_LABEL string_with_blanks [non-null, < 240) chars] Default: Window Manager will supply frame label	Text to place in the frame label of the *executive*'s window. Valid within blocks: W_FRAME
W_TIME_FONT string [non-null, <80) chars] Default: Use default font	The name of a font to use for the time string. Use the 'xlsfonts' command to find a list of available fonts. Valid within blocks: W_CLOCK

Keyword	Description
W_DATE_FONT string [non-null, <80) chars] Default: Use default font	The name of a font to use for the date string. Use the 'xlsfonts' command to find a list of available fonts. Valid within blocks: W_CLOCK
W_GMT_FONT string [non-null, <80) chars] Default: Use default font	The name of a font to use for the GMT string. Use the 'xlsfonts' command to find a list of available fonts. Valid within blocks: W_CLOCK
W_DONT_GRAB Default: Grab control of system console	Don't take control of the system console, just use the *executive*'s console window for messages from the *executive* and its children. Valid within blocks: W_CONSOLE

Action Sets

Action sets, which may be defined either explicitly or implicitly, define a group of actions that the *executive* will perform, in the order they are defined, as a result of either:

1. a button (not associated with a pulldown menu) being selected
2. a menu item being selected
3. a process dying
4. an application process sending a request message to the *executive* (see X_PROC_EXEC_KEY above).

An explicit definition consists of the keyword A_DEFN followed by one or more of the other A_* keywords (each of which may appear more than once, in any order). In this case the A_DEFN keyword defines the name (or tag) of the action set. This tag is used to refer to this action set by the P_ACTION, M_ACTION, or MI_ACTION keywords, or in a message from an application process (see X_PROC_EXEC_KEY above).

An implicit definition consists of one or more of the A_* keywords (except not the A_DEFN keyword) appearing within a P_DEFN block, an M_DEFN block (only if the block defines a button with no associated menu), or an MI_DEFN block. In this case the action set is only associated with the process, button, or menu item where it was defined. In the explicit case, an action set can be referred to by more than one P_ACTION, M_ACTION, or MI_ACTION keyword.

Thus the A_* keywords (except A_DEFN) may appear within an A_DEFN block, a P_DEFN block, an M_DEFN block (only if the block defines a button with no associated menu), or an MI_DEFN block.

Actions within an action set can be performed conditionally based on the setting of flags defined in the configuration file. Up to 30 flags may be defined in the configuration file. Each flag is referred to by a name of up to 30 alphanumeric characters. Each flag can take on the values TRUE or FALSE. Flags can be set to TRUE or FALSE, to the value of another flag, or to the opposite of the value of another flag. Flags are used on IF-ELSE-ENDIF blocks to control whether or not the actions within the block are performed. IF-ELSE-ENDIF blocks can be nested up to 10 levels deep.

There is one special action set where the A_DEFN tag is "a_frame_quit". It is executed when the user selects "Quit" from the *executive*'s Frame Menu. It is likely to contain the A_LOGOUT action, and perhaps some other actions. If no such "a_frame_quit" action set is defined in the configuration file, then the *executive* will perform an A_LOGOUT action when "Quit" is selected from the Frame Menu.

Keyword	Description
A_DEFN string [non-null, < 30 chars] Default: N/A	Defines the beginning of an explicit action set definition, and names the set with the tag.

Keyword	Description
A_CONFIRM string_with_blanks [non-null, < 240 chars] Default: N/A	The string will be displayed in a "Yes/No" popup when this item of the action set is reached. If the user chooses "No", no more actions in the action set will be executed. If the user chooses "Yes", the remainder of the action set will be executed normally.
A_MITEM_ENABLE string [non-null, < 30 chars] Multiple: OK Default: N/A	Tag of menu item to enable (un-grey out).
A_MITEM_DISABLE string [non-null, < 30 chars] Multiple: OK Default: N/A	Tag of menu item to disable (grey).
A_MITEM_REPLACE string [non-null, < 30 chars] string [non-null, < 30 chars] Multiple: OK Default: N/A	First string is tag of menu item to replace, second string is tag of menu item with which to replace. For Motif, you must place these two menu items next to each other in the configuration file or else the replacement will probably do something you don't want.
A_BUTTON_SHOW string [non-null, < 30 chars] Multiple: OK Default: N/A	Tag of menu whose button will be made visible and selectable.
A_BUTTON_GREY string [non-null, < 30 chars] Multiple: OK Default: N/A	Tag of menu whose button will be made visible but unselectable (grey).
A_BUTTON_HIDE string [non-null, < 30 chars] Multiple: OK Default: N/A	Tag of menu whose button will not appear.
A_PROC_RUN string [non-null, < 30 chars] Multiple: OK Default: N/A	Tag of process to run.
A_PROC_RUN_IF string [non-null, < 30 chars] Multiple: OK Default: N/A	Same as A_PROC_RUN, except the process will be run only if it is not currently running.
A_PROC_CMD string [non-null, < 30 chars] string_with_blanks Multiple: OK Default: N/A	First string is tag of process to send command, remainder of line (up to comment delimiter or newline) is command to send to that process via a pipe to its stdin. The process must have been spawned with a pipe to its stdin (this is specified in the P_DEFN block for the process).
A_PROC_CMD_IF string [non-null, < 30 chars] string_with_blanks Multiple: OK Default: N/A	Same as A_PROC_CMD, except if the receiving process isn't running, no error will be logged and the rest of the action set will be executed normally.

Keyword	Description
A_PROC_SIG string [non-null, < 30) chars] string ["SIGINT", "SIGTERM", "SIGKILL", "SIGHUP", "SIGURG", "SIGUSR1", or "SIGUSR2"] Multiple: OK Default: N/A	First string is tag of process to send signal, second string is one of the following signal keywords: SIGINT, SIGTERM, SIGKILL, SIGHUP, SIGURG, SIGUSR1, SIGUSR2.
A_PROC_SIG_IF string [non-null, < 30) chars] string ["SIGINT", "SIGTERM", "SIGKILL", "SIGHUP", "SIGURG", "SIGUSR1", or "SIGUSR2"] Multiple: OK Default: N/A	Same as A_PROC_SIG, except if the receiving process isn't running, no error will be logged and the rest of the action set will be executed normally.
A_MESSAGE string_with_blanks [non-null, < 240 chars] Multiple: OK Default: N/A	String is a message which will be printed to the *executive*'s console window, or if a logger process is running the string will be sent to the logger process instead.
A_SLEEP integer (seconds) [1 <= Value <=30] Multiple: OK Default: N/A	The *executive* sleeps for the requested number of seconds. No menu selections (or child process events) will be handled during the sleep, but they will be handled after the sleep.
A_LOGOUT Default: N/A	Perform the actions associated with "logging out" of the *executive*. If any process is running that is not "P_LOGOUT_OK" (see below), a message is displayed and nothing is done (and any remaining items in the action set are not executed). Otherwise, every running process is killed, and the *executive* exits (after executing any remaining items in the action set).
A_RUN_AT_INIT Default: Don't execute action set at *executive* initialization	Execute this action set at *executive* initialization.
A_RUN_AT_INIT_IF string [non-null, < 240 chars] string [non-null, < 240 chars] optional delimiter "AND" optional string [non-null, < 240 chars] Multiple: OK, but must be together within the block Default: Don't execute action set at *executive* initialization	Same as A_RUN_AT_INIT, except the first string is a subsystem keyword, and the second string is a startup option keyword. These are compared to keywords in the file provided by exeuser_runf_path (). If this subsystem appears in the runfile (from exeuser_runf_path ()) with this startup option, then this action set will be executed at initialization. An additional condition may be specified, if "AND startup_mode" appears after the startup option on the keyword line. If this is specified, then the startup_mode keyword must also match the system startup mode returned by exeuser_runf_mode () in order for this action set to be executed at initialization.
A_SET_FLAG string string [non-null, < 30 chars] Multiple: OK Default: N/A	The first string is the name of a flag to set. The second string can be "TRUE", "FALSE", or the name of a flag preceded by an optional '!' character (no whitespace between the '!' and the first letter of the flag name). The first flag will be set to TRUE, FALSE, the value of the second flag, or the logical NOT of the value of the second flag (if '!' is present). The same flag may be used as the first and second flag arguments (e.g. "A_SET_FLAG Flag1 !Flag1" to invert the value of Flag1).

Keyword	Description
A_DO_IF string [non-null, < 30 chars] Multiple: OK Default: N/A	The string is the name of a flag or the name of a flag preceded by an optional '!' character (no whitespace between the '!' and the first letter of the flag name). The actions in the following block will be performed if the flag is TRUE (if '!' is present they'll be performed if the flag is FALSE).
A_DO_IF_AND string [non-null, < 30 chars] Multiple: OK Default: N/A	The string is the name of a flag or the name of a flag preceded by an optional '!' character (no whitespace between the '!' and the first letter of the flag name). The previous keyword must be an A_DO_IF, A_DO_IF_AND, or A_DO_IF_OR. The association of multiple A_DO_IF_* keywords following an A_DO_IF keyword is as follows (for example): A_DO_IF Flag1 A_DO_IF_AND Flag2 A_DO_IF_OR !Flag3 A_DO_IF_AND Flag4 will be interpreted as: (((Flag1 && Flag2) && !Flag3) || Flag4)
A_DO_IF_OR string [non-null, < 30 chars] Multiple: OK Default: N/A	The string is the name of a flag or the name of a flag preceded by an optional '!' character (no whitespace between the '!' and the first letter of the flag name). The previous keyword must be an A_DO_IF, A_DO_IF_AND, or A_DO_IF_OR. See interpretation comments for A_DO_IF_AND keyword.
A_DO_ELSE Multiple: OK Default: N/A	Indicates beginning of "else" block associated with the previous A_DO_IF if test.
A_DO_ENDIF Multiple: OK Default: N/A	Indicates end of "if" or "if-else" block associated with the previous A_DO_IF if test.

Processes

Each process, including the subexecutive process(es) if needed, is defined by a process definition block, starting with the P_DEFN keyword. This process definition block is referred to in action sets which request that the process be run, or that a command or signal be sent to the process. The P_* keywords must only appear within a P_DEFN block. All A_* keywords except for A_DEFN may appear within a P_DEFN block to define actions to perform whenever the process dies. Note that the A_PROC_RUN keyword (specifying the process itself) may be used to restart the process automatically whenever it dies.

Keyword	Description
P_DEFN string [non-null, < 30 chars] Default: N/A	Defines the beginning of a process definition block, and names the process definition block with the tag.
P_NAME string [non-null, < 20 chars] Default: The process definition tag from the P_DEFN line	Defines the name of the process.
P_PATHNAME string [non-null, < 80 chars] Default: The process name from the P_NAME line	Defines the pathname of the executable file.

Keyword	Description
P_ARGV string_with_blanks [non-null, < 240 chars], at most 20 separate strings Default: The process is started with argc=1, argv[0]=P_NAME	The strings appearing on this line are used for argv[1] through argv[n] when the process is started. Note that the same process can be started with different arguments by creating several P_DEFN blocks that refer to the same process but have different values on the P_ARGV line.
P_PIPES string [one of these: "none", "stdin", "stdout", "both"] Default: none	This defines what pipes to create from this process to the *executive* process which spawns it. If "both" or "stdout" is specified, then stderr will be piped along with stdout. If the process EVER reads stdin, it MUST specify a stdin pipe on the P_PIPES keyword (either "stdin" or "both"). This can't be checked by the *executive* since it has no way to know whether a process will read stdin, so it is up to you to check this. If there is any doubt, specify a stdin pipe just to be safe.
P_UPDOWN_FILE string [non-null, < 80 chars] Default: No such file	The pathname of a file that is to exist if and only if the process is running (the *executive* takes care of creating it and removing it). This provides a simple way for other processes to tell if this process is running. The *executive* will not run such a process if the updown file exists.
P_KEYSTRING string [non-null, <10 chars] Default: Capability disabled	This process wants to receive all messages (from any process) that begin with this string. The entire message (including the keystring) up to and including a newline will be sent to this process.
P_PARENT integer [-1 <= Value <3)] Default: -1	Number of the *executive* process which will spawn this process. A value of -1 means the main *executive* should spawn it. A non-negative value is the index of the subexecutive which should spawn it. The reason to have a subexecutive spawn a process is that if too many processes are spawned with pipes, the main *executive* may run out of file descriptors. Using a subexecutive is just a way to get more file descriptors.
P_PRIVILEGED Default: Process is not privileged	This process is privileged and can only be run if its process name appears in the list of privileged processes returned by exeuser_privs (). Furthermore, the "setuser" process cannot be run if this process is running.
P_LOGOUT_OK Default: Cannot logout with process running	This process may be up at logout, and will then be killed by the *executive*.
P_IAM_LOGGER Default: This is not the logger process	This process is the logger, and will receive messages from all other processes (and will do something reasonable with them, like display them in a text window).
P_IAM_SETUSER Default: This is not the setuser process	This process is the setuser process, which determines which privileged processes are currently eligible to run. It can't be run if any privileged process is running, and no privileged process can run if it is running.
P_SUBEXEC integer [0 <= Value <3] Default: This is not a subexecutive process	This process is a subexecutive process, and the value is its subexecutive index number.
P_ACTION string [non-null, < 30 chars] Default: No action set	The string is the tag (from an A_DEFN line) of an action set to be executed whenever the process dies.

Menus and Buttons

Each menu is defined by a menu definition block, starting with the M_DEFN keyword. The *executive* menus are activated from buttons in the buttons subwindow. For Xview, the button has a little down arrow in it. The menu appears below the button when the right mouse button is pressed and held on the button. For Motif, the menu appears below the button when the left mouse button is pressed and held on the button. The standard Motif keyboard menu traversal capabilities are supported.

It is also possible to define standard pullright menus that are activated from a menu item in a higher level menu.

The items in a menu are defined by MI_DEFN blocks appearing after the M_DEFN block. All MI_DEFN menu items after an M_DEFN menu and before the next M_DEFN menu will belong to the preceding M_DEFN menu.

It is possible to define a button that does not activate a menu but that causes an action set to be executed. To do this, just create an M_DEFN block that is not followed by any MI_DEFN blocks. Such an M_DEFN block may contain A_* and M_* keywords (except it may not contain the M_PULLRIGHT keyword).

Keyword	Description
M_DEFN string [non-null, < 30 chars] Default: N/A	Defines the beginning of a menu definition block, and names the menu with the tag.
M_BUTTON_LABEL string_with_blanks [non-null, < 40 chars] Default: None	The label for the menu's button. For Motif, it may contain newlines to create a multi-line label.
M_BUTTON_X integer (pixels) [>=0] Default: Use default positioning	The x coordinate of the left edge of the menu's button in the *executive*'s button panel.
M_BUTTON_Y integer (pixels) [>=0] Default: Use default positioning	The y coordinate of the top edge of the menu's button in the *executive*'s button panel.
M_DISABLED Default: Menu (and button) are enabled	The menu (and button) are initially disabled. The button will not be visible. Its place in the button panel will be held if M_SAVE_SPOT is specified.
M_DISABLED_IF string [non-null, < 240 chars] string [non-null, < 240 chars] optional delimiter "AND", optional string [non-null, < 240 chars] Multiple: OK, but must be together within the block Default: Don't disable menu (and button)	The first string is a subsystem keyword, and the second string is a startup option keyword. These are compared to keywords in the file provided by exeuser_runf_path (). If this subsystem appears in the runfile (from exeuser_runf_path ()) with this startup option, then this menu will be disabled at initialization. An additional condition may be specified, if "AND startup_mode" appears after the startup option on the keyword line. If this is specified, then the startup_mode keyword must also match the system startup mode returned by exeuser_runf_mode () in order for this menu to be disabled at initialization. Its place in the button panel will be held if M_SAVE_SPOT is specified.
M_SAVE_SPOT Default: Don't save spot for button if menu and button are disabled	If M_DISABLED is specified, or if M_DISABLED_IF is applicable, then save a spot for the menu's button in the button panel.

Keyword	Description
M_PULLRIGHT Default: Menu is a pulldown from a button	Not allowed if the menu definition block defines a button with no associated menu. This menu is a pullright menu off of a menu item in another menu (or several menu items, perhaps) (see MI_PULLRIGHT). Therefore no button will be created for it.
M_ACTION string [non-null, < 30 chars] Default: No action set	Only allowed if the menu definition block defines a button with no associated menu. The string is the tag (from an A_DEFN line) of an action set to be executed whenever the button is selected.

Menus Items

Each menu item is defined by a menu item definition block, startin with the MI_DEFN keyword. The items in a menu are defined by MI_DEFN block appearing after an M_DEFN block. All MI_DEFN menu items after an M_DEF menu and before the next M_DEFN menu will belong to the first M_DEFN menu The MI_* keywords must only appear within an MI_DEFN block. All A_ keywords except for A_DEFN may appear within an MI_DEFN block to defin actions to perform whenever the menu item is selected.

Keyword	Description
MI_DEFN string [non-null, < 30 chars] Default: N/A	Defines the beginning of a menu item definition block, and names the menu item with the tag.
MI_LABEL string_with_blanks [non-null, < 40 chars] Default: None	The label for the menu item in the menu. For Motif, it may contain newlines to create a multi-line label.
MI_DISABLED Default: Menu item is enabled	The menu item is initially disabled. The menu item will appear in the menu greyed out, unless MI_DONT_DISP is specified, in which case it will not appear in the menu at all.
MI_DISABLED_IF string [non-null, < 240 chars] string [non-null, < 240 chars] optional delimiter "AND", optional string [non-null, < 240 chars] Multiple: OK, but must be together within the block Default: Don't disable menu item	The first string is a subsystem keyword, and the second string is a startup option keyword. These are compared to keywords in the file provided by exeuser_runf_path (). If this subsystem appears in the runfile (from exeuser_runf_path ()) with this startup option, then this menu item will be disabled at initialization. An additional condition may be specified, if "AND startup_mode" appears after the startup option on the keyword line. If this is specified, then the startup_mode keyword must also match the system startup mode returned by exeuser_runf_mode () in order for this menu item to be disabled at initialization. The menu item will appear in the menu greyed out, unless MI_DONT_DISP is specified, in which case it will not appear in the menu at all.
MI_DONT_DISP Default: Display menu item greyed out if it is disabled	If MI_DISABLED is specified, or if MI_DISABLED_IF is applicable, then don't display the menu item at all.
MI_HIDDEN Default: Menu item is displayable	Do not put this menu item in the menu, but create the menu item and save it for later. It may be used later in an A_MITEM_REPLACE action to replace another menu item.
MI_PULLRIGHT string [non-null, < 30 chars] Default: This menu item does not have a pullright menu	The menu whose tag matches the string is a pullright menu off of this menu item.

Keyword	Description
MI_PRIV_PROC string [non-null, < 30 chars] Default: Menu item is not associated with any process	Tag of a privileged process. If this process is not currently privileged to run, then this menu item will be greyed out.
MI_ACTION string [non-null, < 30 chars] Default: No action set	The string is the tag (from an A_DEFN line) of an action set to be executed whenever the menu item is selected.
MI_SEPARATOR Default: No separator appears above the menu item	Place a separator widget above this menu item.